Problem Oriented Engineering for Software Safety

Thesis

Problem Oriented Engineering for Software Safety

D. P. Mannering B.Sc., M.Sc.

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science

Department of Computing

Faculty of Mathematics and Computing

The Open University

June 2009

Contents

List of Figures

List of Tables

Author's Declaration

Some of the material in this thesis appears in the following papers.

- J. G. Hall, D. Mannering, and L. Rapanotti. Arguing safety with Problem Oriented Software Engineering. In HASE'07, pages 23–32. IEEE, 2007.

- D. Mannering. Implementable Requirements in Problem Orientation. In IWAAPF '08. ACM, 2008.

- D. Mannering, J. G. Hall, and L. Rapanotti. Safety process improvement: Early analysis and justification. In Proceedings of the IET Second International Conference on System Safety 2007. IET, 2007.

- D. Mannering, J. G. Hall, and L. Rapanotti. Safety process improvement with pose and alloy. In Proceedings of The 26th International Conference on Computer Safety, Reliability and Security (Safecomp 2007). Springer, 2007.

- D. Mannering, J. G. Hall, and L. Rapanotti. Towards Normal Design for Safety-Critical Systems. In Proceedings of FASE 2007, pages 398–411, 2007.

- D. Mannering, J. G. Hall, and L. Rapanotti. Safety Process Improvement: Using POSE and Alloy. In Safety System Symposium SSS'08, pages 398–411, Bristol, England, February 2008.

I declare that no part of this material has previously been submitted to a degree or any other qualification at this University or another institution.

I further declare that this thesis is my original work, except for clearly indicated sections where appropriate attributions and acknowledgements are given to work by other authors.

Derek Mannering

Acknowledgements

Thanks are due to the following individuals and organisations without whose assistance this thesis would not have been possible.

I am especially grateful to my supervisors Jon Hall and Lucia Rapanotti without whose help this thesis would not have been possible. They have shown patience, and freely given advice, guidance and knowledge. Particular qualities of merit are Jon for his precision and Lucia for her focus – both qualities which have been instrumental in forming this work.

I am also grateful to others at the Open University who have given useful support and encouragement including Marian Petre and Bashar Nuseibeh.

My work colleagues have provided invaluable benefit, help and shown considerable patience whilst I have been undertaking this research. Particular thanks are due to my colleague and manager Phil Williams for his encouragement and understanding, and also to Andy Fry for his helpful comments.

Finally I would like to say a very big thanks to my family who have given support, help and suffered much in the last few years as weekends and evenings have "disappeared" whilst I have pursued my research objectives. My wife Karen, my children Georgia and Lydia, and my parents Derek and Jean have all done their bit in supporting me through this sojourn. The last words are a dedication in memory of my nephew Richard - thesis successfully completed as promised.

Abstract

Safety critical systems must satisfy stringent safety standards and there development requires the use of specialist safe software system development (SSSD) approaches as the complexity and penetration of these systems increases. These SSSD approaches satisfy certain useful properties that make them suitable for safety system development. The first objective of this thesis is to select a candidate SSSD approach and evaluate its capabilities against a set of useful properties identified from reviewing a group of existing SSSD approaches, and thus show that this candidate SSSD approach is appropriate for use in safety system development.

In addition, a second objective is to use this candidate SSSD approach to improve the early life cycle phase of an existing industrial safety development process used to develop embedded avionics applications. In particular to allow issues to be resolved earlier in the development, which are currently not being uncovered until much later in the development when they are much more difficult and expensive to correct. This involved the identification of further properties and issues that the candidate SSSD approach must address.

The overall aim is to demonstrate that this candidate SSSD approach can be used in the early phase of a safety system development to derive a validated specification that can be subjected to safety analysis to show that it satisfies the identified system safety properties and thus forms a viable basis for the rest of the development.

Chapter 1

Introduction

Embedded computer systems are becoming more pervasive [64, 57] and their scope and complexity is ever increasing [90]. These are used in areas such as avionics, rail and process control and more recently in medical applications [64]. The harm potential of such systems is recognised: the outcry associated with a major rail accident, air crash or nuclear power incident demonstrates the importance of safety as a significant non-functional requirement [96]. This concern has been expressed in a variety of national and international safety standards such as the UK MoD's Defence Standard (DS) 00-56 [125] and the international IEC 61508 [51], and the production of safety critical and safety related systems is increasingly being regulated and controlled. All the major safety standards are concerned with the management and mitigation of risk as a key element. Risk is managed through the concept of a safety life cycle [125, 51, 28], and although terminology differs the basic sequence of hazard identification, hazard analysis and risk mitigation is followed by all [65]. Further, this safety life cycle must run in parallel to and be integrated with the design and development life-cycle. That is, the production of a design artefact is associated with a corresponding safety analysis task. The goal is to ensure that safety is not a "bolt-on" extra performed when the design is finished, but that it has appropriate influence on the design decisions as the system development progresses

[66, 30, 69]. It follows that there is a need for safe software system development (SSSD) approaches that integrate the safety and development life cycles to manage the safety risks.

1.1 Issues in Current Safety Engineering

The safety standards require hazard identification and preliminary hazard analysis to occur in the early phases of the development process (e.g., [89]). This is consistent with studies that have shown that a large proportion of anomalies occur at the requirements and specification stages of a system development [29, 69]. A study by Lutz concluded that safety-related software errors arose most often from inadequate or misunderstood requirements [79], indicating the need for a more careful analysis of the requirements to ensure adequate assimilation and comprehension of them. Further, other work has highlighted the need to conduct a safety analysis of the requirements [26, 33]. These factors all support the notion that safety must be built into the design, and that the evolving design representations analysed to demonstrate that they have the desired safety properties [70] as early as possible in the life cycle - preferably during the requirements phase. One of the objectives of this work is to propose an approach able to support safety analysis within the early phases of requirements analysis and high-level architectural design.

A direct result of the increasing use and penetration of software intensive systems is the increase in system complexity which it allows [64]. More things can be done with a software system - so they are [19]. In certain sectors system complexity is rising exponentially [91]. There is a need to cope with this increased complexity as part of the development process, and this is often achieved by using reduction techniques [69] and/or abstraction to structure the problem [71]. The complex interaction between diverse system elements (e.g., operators and computer control system) can result in unsafe behaviour due to unexpected emergent properties [72].

As a result increased complexity can arise in a number of forms which must be covered by an adequate safety analysis approach. Another objective of this work is to propose concept tools to deal with the complexity of analysing safety problems and synthesising adequate safety solutions.

Formal specifications have been a focus of software engineering research for many years and have been applied in a wide variety of settings [126], including safety critical applications [80]. Formal specification is required by a number of safety [51] and security [15] standards for their highest integrity applications. The use of such techniques is not universal, but to some they provide a "gold standard" [43] of what can be achieved. However researchers have noted that the perceived benefits of applying formal specification techniques are not being realised [44, 126]. This has been attributed to a number of causes including no methodological guidance for using formal techniques [44]; formal techniques are not well integrated with the system analysis phase - there is a need to analyse the system and its environment before producing the specification [44, 57]; and many of the techniques have poor separation of concerns in that the desired behaviour of the environment is not well distinguished from the extant properties of the environment [126].

As an approach, some have advocated extending the scope of formal techniques within the structure of an integrated RE process based on formality covering elicitation, modelling and analysis, communication and validation, and also stressing the importance of the environment [128][44]. This has been successful in extending the reach of formal methods, but has not overcome the disconnect of Turski's problem [120]: that real-world domains are not necessarily expressible in any single linguistic system; and that the notion of mathematical (logical) proof does not apply to them. This disconnect manifests itself for formal methods in the following ways: they do not adequately bridge the gap between the uncertainty of the real-world and the formality of the formal world; they do not distinguish between the indicative properties of the environment and the optative properties of the requirements; they

neglect the validity aspects of the requirements; and they are not good at validating progress towards the solution. Formal models are precise entities whose properties can be demonstrated by proof. The real world is more variable, and the formal models produced are only approximations to it. Yet another objective of this work is to address Turski's divide and propose an approach in which formality is properly situated within safe engineering practice.

Engineering is different from science and mathematics [82], although the techniques and approaches used in both science and mathematics can be used in an engineering context. In much simplified terms, engineering is concerned with building artefacts which transform the physical world [82], whilst science is concerned with universal properties of the universe and mathematics universal truths [54]. In fact, engineering tools are most useful when they are specific to a particular problem, and different classes of problem may demand different languages to represent them [82]. Further, software engineering itself covers a broad range of quite different application areas, which are best served by specific problem oriented language notations [121].

If software engineering (SE) is to attain the benefits achieved by other engineering professions such as aeronautical engineering, then it must seek to satisfy the "normal design" principles and where possible avoid radical ones [82, 55]. Vincenti [131] defines "normal design" as what the engineer is engaged in when s/he knows from the outset

> "how the device in question works, what are its customary features, and that, if properly designed along such lines, it has a good likelihood of accomplishing the desired task."

Much of the routine design encountered in traditional engineering disciplines works *normally*. Some have recently observed that software engineering does not: Maibaum [82] states that

> "SE ignores the principles of engineering design and almost always adopts radical design methods, to its detriment."

Jackson states [55]

> "Though less conspicuous than radical design, normal design makes up by far the bulk of day-to-day engineering enterprise. Unfortunately, this is not true of software engineering."

To write useful software it is not enough to deal with just the computer and its software, one must also deal with the complexities of the natural or "real" world in which the application resides – and the gap between the specification and the requirements phenomena must be bridged by the problem domain properties (domain knowledge) [54]. The importance of domain knowledge is identified by a number of authors [54, 82, 121, 127]. Further, Maibaum [82] identifies systemising domain knowledge and the use of architecture to encapsulate design choices as two key requirements for achieving *normality* in design. A key characteristic of the approach we propose is a clear separation of concerns between the system under design, its desirable properties and the extant properties of its deployment context.

1.2 Aim, Objectives and Method

The first objective of this thesis is to propose and evaluate an SSSD approach for safe software system engineering able to meet the following specific goals:

- integrate safety analysis within the early phases of development, particularly requirement analysis and high level architectural design
- allow for both formal and informal safety analysis techniques to take place
- capture and separate contextual knowledge from the design of the system and its desirable properties

- keep track of safety artefacts for validation and certification purposes

- integrate with current safety engineering practice.

The approach we propose is based on problem orientation [133], and particularly, their embodiment in the Problem Oriented Software Engineering (POSE) framework [41, 39] whose particular choice is argued and justified in Chapter 2.

The second objective is to improve the early phases of an existing safety development process. The work is situated within the safety development process in use in the author's company; this process is tailored to the production of embedded safety systems with a significant software content and has been used to produce a number of safety critical avionics systems that have been accepted into service. The process is representative of the industry, and is generally acknowledged to be successful and robust. However, a number of issues have been identified with it, particularly in the early stages of the development. One of the major drivers for this research was to investigate how to address these issues. As well as providing a useful driver, situating the work within a practical industrial process is also beneficial in the evaluation of the research outcome, whose impact can be measured directly through industrial application and case studies.

Therefore the method used in this research is a combination of literature survey, problem analysis and the use of case studies. In more detail, the role of the literature survey is to:

- identify the underlying safety principles (the safety context) that impact the research

- identify issues and properties associated with the early phases of a safety development process

- evaluate existing safety analysis and safe software system development approaches

- use this information to produce a set of generic properties that our proposed engineering approach should possess, and against which it can be evaluated.

Problem analysis is performed within the context of the author's company development process and practice, with a view to identify additional specific properties and issues that the candidate engineering approach should also be able to address. The ensuing research work based on case studies is used to evaluate the proposed approach against the established metrics. The overall aim is to demonstrate that POSE, and its extensions developed by this work, can be used in the early phase of a safety system development to derive a validated specification that can be subjected to safety analysis to show that it satisfies the identified system safety properties and thus forms a viable basis for the rest of the development.

1.3 Thesis Structure

The thesis is organised as follows.

The initial task is to scope and investigate the safety context and this is addressed in the early part of Chapter 2. This will consider the impact of safety standards, the use of safety terminology, the safety life cycle, safety analysis techniques and making safety case arguments. A number of approaches that have been successfully used to develop safety systems are then introduced and used later in the chapter to identify a set of properties that an appropriate safety engineering approach should possess. The chapter concludes with the justified selection of POSE as the safety engineering approach.

Chapter 3 introduces the early phases of an industrial safety development process used by the author's company to develop embedded safety systems. Some problems that have been identified with this process are discussed and these are summarised into a number of properties and issues which are used in conjunction with the properties identified in Chapter 2 to evaluate POSE in the remainder of

the thesis.

Chapter 4 introduces POSE in its basic, "vanilla" form through its application to an avionics Decoy Controller case study. Its capabilities in dealing with the identified properties and issues are evaluated and its performance recorded in tabular form. The identified shortfalls provides a focus and target for the major research work that is reported in the chapters that follow.

The goal of Chapter 5 is to develop an algorithm for the POSE Problem Progression Transformation (PPT) that supports its application to allow requirements to be transformed such that the specification of the system can be derived directly. An extension of the Decoy Controller case study introduced in Chapter 4 is used to illustrate the ideas involved and to demonstrate the successful application of the PPT. This chapter also covers the development of a safety analysis process which exploits POSE's characterisation of software problem to achieve an adequate and effective safety analysis. Next, a Stores Management System (SMS) case study is introduced that demonstrates how POSE can be used in conjunction with formal development techniques.

The goal of Chapter 6 is to address the remaining identified issue of early safety analysis. The POSE safety pattern is introduced (section 6.1) to achieve this goal and its utility demonstrated through its application to a third, Audio Warning System, case study. The case study, presented in section 6.3, provides evidence to support the usefulness of all the POSE extensions developed in the earlier chapters and demonstrates that they successfully address all properties and issues identified.

The thesis finishes with a detailed discussion and conclusions of the results obtained in Chapter 7 and possible future work is covered in Chapter 8. The appendices provide additional information that supports aspects of the research work, which are referenced from the main text as appropriate.

Chapter 2

Literature Review

The aim of this chapter is to introduce and discuss in detail the background and basis for the research into problem orientation and in particular the use of Problem Oriented Software Engineering (POSE) to build "safe software". The chapter begins with a discussion of the problems associated with the different definitions that abound in safety engineering and identifies the nomenclature used in this work. Then the important areas of system safety and system safety analysis are reviewed, together with a number of development approaches used to produce safety software, followed by a discussion of problem orientation. An analysis is performed in section 2.5 to identify ten useful properties that a safety development approach should possess which are used to select a candidate approach – POSE – for use in the remainder of this thesis. The chapter ends with a summary.

2.1 Nomenclature

The term *requirement* [53, 127, 96] covers a number of levels varying from entities at the high level such as the users required behaviour (functional requirements) and dependability requirements (e.g. safety and security) through to lower level entities such as specific requirements allocated to the hardware or software. In this

document the term requirement or system requirement is used to refer to the high level entities unless explicitly stated. For example, in the solution of the entailment sequent $W, S \vdash R$ used extensively in this work, the term R is the overall system requirement, whilst S represents the specification of the processing machine. In effect, S represents the allocated software requirement for the machine, but will be called the specification or machine specification in this document.

Similarly, it is also worth noting that the term *architecture* [103, 17, 34] also occurs at a number of levels covering system architecture, software architecture and hardware architecture. As this work is concerned with the early phases of a development it is the system architecture that is the main focus. However the prefix of system, software or hardware will be used where this aids comprehension.

This work is concerned with software safety with respect to embedded real-time systems. Actually the term software safety is something of a misnomer, software by itself cannot cause harm. It is only in the context of a system operating in a potentially unsafe environment that "unsafe" software can result in a safety problem [93]. For example, an aircraft flight control system operating on a ground-based test rig has far different safety implications to that same flight control system being used to control a fast jet in low level flight operations. That is, software safety is a property requiring software and a potentially unsafe context. In this work the term *software safety* will be used as an abbreviation for the safety aspects of the software operating in its system context that in turn, is operating in the overall system environment.

Moreover, there is no agreed definition of the terms used in the description of a potentially unsafe environment; in particular, there are differences between the common usage of the terms *failure*, *fault* and *error* in some software developments and common usage in traditional safety engineering [69]. However, analysis of the definitions used in IEC 61508 [51], DS 00-56 [125] and by Leveson in her book Safeware [69] does show a good correlation for the important terms *harm*, *hazard*,

Term	Source	Safety Term Definition
Harm	IEC 61508	Physical injury or damage to the health of people either directly or indirectly as a result of damage to property or to the environment
	DS 00-56	Same as IEC 61508
	Leveson	*** Not Used *** (Uses Loss)
Hazard	IEC 61508	Potential source of harm
	DS 00-56	A physical situation or state of a system, often following from some initiating event, that may lead to an accident
	Leveson	A state or set of conditions that contributes to an accident
Risk	IEC 61508	Combination of probability of occurrence and the severity of that harm
	DS 00-56	Combination of the likelihood of harm and the severity of that harm
	Leveson	Hazard severity, likelihood and exposure
Accident	IEC 61508	*** Not Used ***
	DS 00-56	An unintended event, or sequence of events, that causes harm
	Leveson	Undesired and unplanned event that results in a loss

Table 2.1: Safety Terminology Comparison

risk and *accident* as shown in Table 2.1. Inspection shows that IEC 61508 and DS 00-56 agree on the term *harm* and all three agree on the definition of *risk*. Leveson [69] uses *loss* in place of *harm*, but allowing for this it can be seen that DS 00-56 and Leveson have similar definitions for *accident*. IEC 61508 relates a *hazard* directly to *harm*, whereas both DS 00-56 and Leveson use *accident* as an intermediate part of the definition.

IEC 61508 and DS 00-56 both define *safety* in terms of *risk* acceptability. In contrast, Leveson defines *safety* as freedom from *accident* or *loss*. Using a risk-based definition for safety is more useful from an engineering viewpoint since absolute safety cannot be achieved. Most modern safety standards are based on the principle of reducing risk to acceptable levels [8], accepting the fact that absolute safety (zero risk) cannot be achieved [105, 90]. For example, IEC 61508 requires the identification of the risks posed by the system under development - and that any such risks will be reduced to tolerable levels by including safety functions in the design responsible

for providing the necessary risk reduction. Therefore risk management is an integral part of modern safety system development. IEC 61508 and Leveson do disagree on the relative status of *failure* and *fault*, but are similar for *error*. DS 00-56 has the same definition for error as IEC 61508. This text will use the definitions from the international standard IEC 61508 supplemented with the use of *accident* (as in DS 00-56). However, the text will avoid the use of the term *fault* as there is less agreement and more confusion concerning its definition in different texts. This means that a *failure* is the

> "termination of the ability of a functional unit to perform a required function",

and an *error* is

> "discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition".

The latter is similar to the Leveson definition of *error* which is "design flaw or deviation from desired or intended state".

2.2 System Safety

According to Leveson [69]:

> "The primary concern of system safety is the management of hazards: their identification, evaluation, elimination and control through analysis, design and management procedures"

System Safety is governed by a variety of national and international safety standards such as the UK MoD's Defence Standard (DS) 00-56 [125] and the international IEC 61508 [51]. All the major safety standards contain the notion of a safety life cycle [125], [51], [28], [109], and although terminology differs the basic sequence

of hazard identification, hazard analysis and risk mitigation is followed by all [65]. As noted at the beginning of Chapter 1, this safety life cycle must be integrated with the design and development life cycle. The safety life cycle expects comprehensive hazard identification and safety analyses to demonstrate that the implemented system is adequately safe with respect to the system's integrity requirements.

The safety standards are backed up by a legal framework that differs from country to country [65]. For example in the UK the legal framework includes the Health & Safety at Work Act (HSWA) 1974, The Management of Health and Safety at Work Regulations (MHSW Regs), contract law (Tort), Sale of Goods act and other EU derived legislation. In particular, the ALARP (As Low As Reasonably Practicable) principle [29] is laid down in law within the HSWA and is an important aspect of IEC61508 [51] and DS 00-56 [125]. ALARP provides high level adequacy criteria for determining when you have done enough, although it is not so easy to apply in practice, since it is difficult to define what constitutes *enough*. Therefore, good practice and legal necessity combine to demand that an appropriate (as defined above) safety life cycle should be followed, and this means that hazard identification is a key task that must be performed at a suitable early point in the development process. In DS 00-56/2 [1], [123], the early tasks of the safety life cycle (hazard identification, preliminary analysis and initial risk estimation) are collected together into the Preliminary Hazard Analysis (PHA) phase. Leveson also uses PHA as the initial safety life cycle phase [69]. Other standards [28], [109] use a different terminology, but the task is essentially the same. This research uses the term "Preliminary Safety Analysis" (PSA) to cover the specific early safety analysis phase it wishes to cover – the term PSA (refer to section 4.4) being selected so it would not be confused with the terminology used by the different standards and guidelines.

[1] Version 2 of DS 00-56 contained a full safety life cycle description, later versions of DS 00-56 adopted a more objective-based stance, and relegated the safety life cycle details to the guidance text.

2.2.1 Safety Cases

A safety case [125, 62, 12] is a documented body of evidence providing a compelling, comprehensive and valid argument that a system is adequately safe for a given application in a given environment. Safety standard DS 00-56 [125] mandates that a safety case should address:

- the management of risk commensurate with the potential risk posed by the system and its complexity;
- the validity of the safety requirements, i.e., that they are derived through thorough analysis and are traceable with respect to the system as designed and implemented, together with evidence of their satisfaction;
- the well-foundedness of assumptions about the system, its operating environment or modes of use upon which the safety argument is based, with a justification that such assumptions are realistic and reasonable.

Ideally the safety case should be made about the product as product-based evidence is more compelling [93]. However, there is still a need for process evidence to cover the system configuration, and the fact that it is often difficult to provide enough direct supporting evidence for a product, and process evidence has to be used.

The UK safety industry has moved towards the widespread use of safety cases to present the safety argument for system safety, since the Cullen Report [22] on the Piper Alpha disaster. The trend [61] is that recent safety standards such as DS 00-56 Version 4 [125] and CAP 670 [3] have adopted a goal based approach for the presentation of a cogent argument backed by adequate evidence that the developed system is safe for its intended use. The argument is required to show that the safety risks associated with deploying the developed system have been reduced to a tolerable level. Therefore risk management is an integral part of safety

system development. Early safety cases were very large monolithic documents that were difficult to comprehend, this resulted in the development of goal structuring notation (GSN) to present the safety arguments in a more accessible, systematic form [59]. GSN is a graphical means of showing the structure of the overall safety argument, and has become the de facto standard means for presenting safety cases for UK military systems.

In GSN goals represent statements that must be true for the safety argument to hold. One useful strategy is to divide the target goal into a number of sub-goals that are individually addressed. Eventually, the process of refining produces goals that are sufficiently detailed to be supported directly by appropriate evidence (these are termed leaf goals), in which case a solution is used to designate the evidence. *Contexts* are used in conjunction with the *Assumptions* to provide a description of the circumstances and constraints under which the safety case is valid. They can also be used to provide explanations of the terms used in the other constructs. *Assumptions* are used to document statements that are taken to be self-evidently true in the safety case.

Even with goal structures there remain acknowledged difficulties in the construction of safety cases, and here we adopt the characterisation from [35]:

- The difficulty of Disparate Descriptions: that of combining disparate pieces of the evidence, such as narrative, requirements, claims, plans, activities or goals [14, 132]; and

- The difficulty of Post-Hoc Assurance: that, traditionally, safety cases are developed post-design and testing with known drawbacks including: expensive redesign when the current design is indefensible; expensive system over-engineering so that the design can be defended; and loss of the rationale for the safety aspects of the design [60].

The later chapters in this thesis will show how POSE can be used to mitigate these difficulties.

2.2.2 System Safety Analysis

The safety standards expect appropriate safety analysis tasks to be applied early in the life cycle, iteratively and on an ongoing basis. Generally this is taken to mean it should occur during the requirements capture and high level specification phases (e.g., see [89], [110]). As noted at the beginning of Chapter 1, this is consistent with the studies that have shown that a large proportion of anomalies occurs at the requirements and specification stages of a system development [29], [69]. Further, investigations have shown that the anomalies of interest are not restricted to just component or function failures, but include significant contributions from factors that are emergent properties of the interactions of complex systems [69], [98]. The abstract descriptions available in the early phases of the development support the need for a comprehensive analysis of the interactions between the system components, operators and environment - an analysis that becomes much more difficult as the amount of design detail increases. Therefore it would be useful if the PSA consisted of the requirements model being subjected to a comprehensive safety analysis to demonstrate that it can satisfy the required system integrity requirements (including emergent properties), and hence form an adequate basis for the ensuing development [33]. So, there is evidence that supports the notion that the safety analysis should be applied to a suitable representation of the requirements so that its emergent safety and failure behaviour can be investigated.

There are many techniques that underpin modern safety analysis and each technique has its advantages and disadvantages. The most commonly used techniques are inductive techniques such as Functional Failure Analysis (FFA) [109, 92], HAZOPS [6, 106] and Failure Modes and Effect Analysis (FMEA) [95, 27], and deduc-

tive techniques such as Fault Tree Analysis (FTA) [130]. However, they all share a number of common properties. For example, the success of these techniques depends directly on having an appropriate representation model that must be validated, and the validation must cover mis-representation of information [1]. Basically if the model does not include the feature of interest then the analysis is unlikely to capture it. They are systematic, but rely on informal reasoning. This means they can be applied to a large class of representations both formal and informal (e.g. [30, 112]). However, establishing the efficacy and rigour of the results obtained is less easy than for example, those results obtained from formal proof or model checking [99].

Attempts have been made to enhance these techniques specifically for use with embedded real-time systems. For example [106] with HAZOPS, Software Failure Modes and Effects Criticality Analysis (SFMECA) [81], Software Fault Tree Analysis (SFTA) [81] and SHARD [102] have all been used successfully, can be integrated and used to provide productive results. However, a number of significant issues are raised when applying conventional safety analysis techniques to software based systems [107]. These techniques are manual, so depend on human understanding, which can be limited for a complex software system; if a formal model exists then it is possible to harness the power of formal analysis and proof techniques to demonstrate desirable properties [4, 119], by using formal analysis in place of manual (albeit systematic) analysis. For example, a technique called Software Deviation Analysis (SDA) can be applied to an RSML model [107].

There is a consensus (e.g., see [110, 112, 125]) that inductive techniques such as FFA, HAZOPS and FMEA, should be used for the identification of hazards; whilst a deductive technique, such as FTA, should be used for finding the causes of and contributors to an identified hazard. Many current approaches use a single technique, often combined with a checklist based on previous experience. This has its problems as no single technique is ideal for all the required hazard identification and analysis tasks, and therefore is not the most appropriate way to proceed at

such a crucial phase (Leveson notes that 70% - 90% of safety related decisions are made in the concept phase [72]). Ways of combining the capabilities of techniques for safety analysis form one of the objectives of pursuing this research.

2.3 Safe Software System Development Approaches

A number of development approaches have been specifically targeted at safety related systems, whilst others have included safety as a significant element. The important safe software system development (SSSD) approaches with significance to this work are summarised in the following, and will be further analysed in section 2.5.

2.3.1 Software Cost Reduction

The Software Cost Reduction (SCR) method follows a four step process for constructing a requirements specification that is based on the concepts defined in the Parnas 4-Variable Model [19, 100] – including the notions of monitored and controlled quantities, and the relations *NAT* and *REQ*. An SCR form of this model, from [9], is shown in figure 2.1. The first step creates an idealised SRS (Systems Requirement Specification). The desired system behaviour is documented in the SRS by describing the relations NAT and REQ, where NAT represents constraints imposed by the environment (such as physical laws) and REQ defines the idealised required behaviour in terms of the monitored and controlled quantities - these are M and C respectively in Figure 2.1. The SRS describes the desired behaviour in terms of the system interacting with its environment, and does not include implementation detail (hence the term *idealised*). This demonstrates that the SCR process is consistent with [133], and concentrates on *WHAT* is required, not *HOW* it is to be achieved. The ensuing phases cover the *HOW* aspects, by refining this idealised model by successively adding design detail.

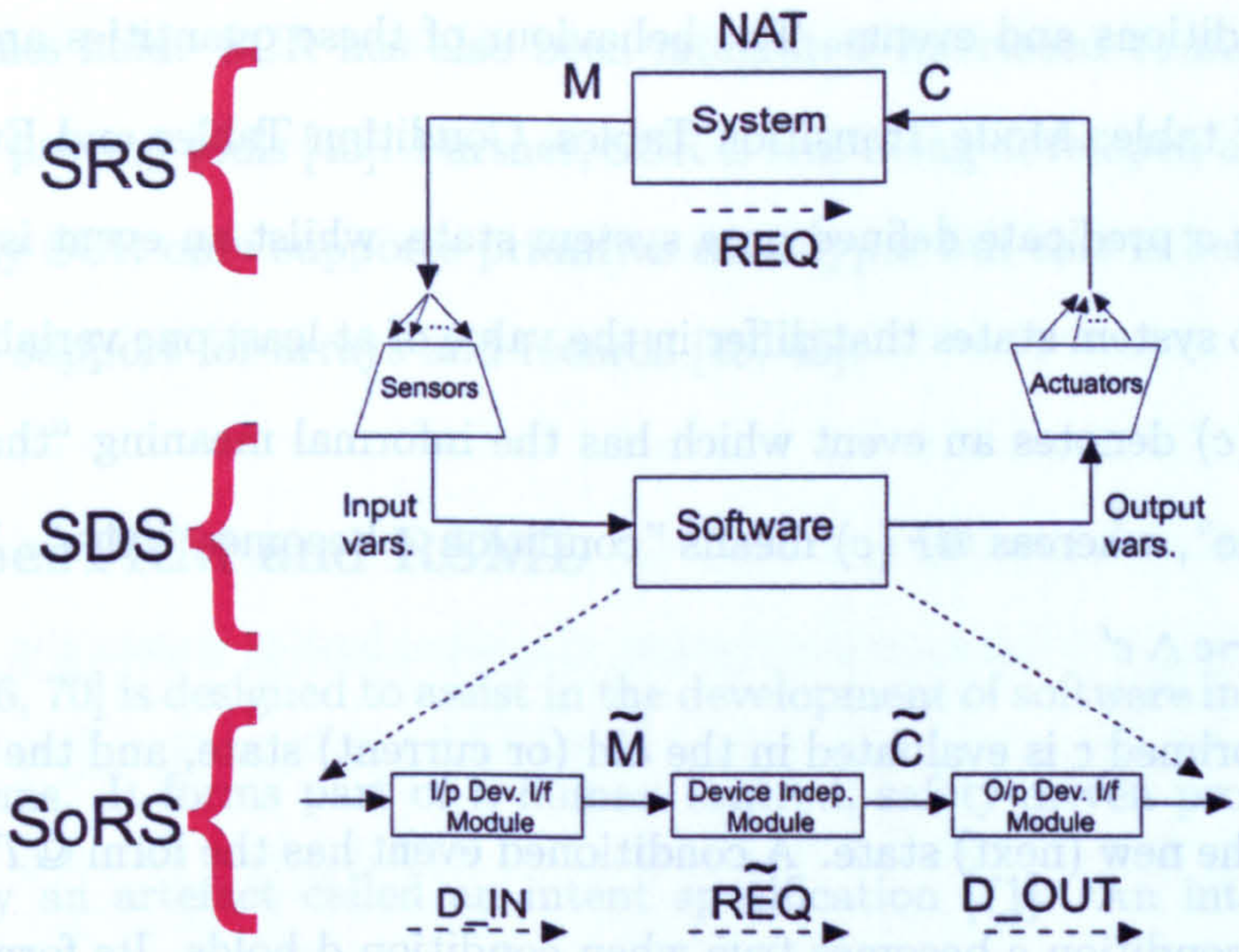

Figure 2.1: SCR Four-Step Process (Based on Parnas 4-Variable Model)

The SDS (System Design Specification) is developed in the second step. This introduces the system's input and output devices (e.g., sensors for monitored and actuators for controlled quantities). Step 3 creates the SoRS (Software Requirements Specification) which refines the SDS by adding input/output design detail. This is achieved by adding modules which use values read from input devices to calculate values of the monitored quantities (D_IN) and which use the computed values of controlled quantities to drive output devices (D_OUT). The Input Device Interface Module (D_IN) takes the input variables output by the sensors and produces an estimate of the monitored quantities, M. This means that the Device Independent Module can produce an estimate for REQ, which mimics the REQ at the overall system level and hence its requirements are those defined in the SRS. It produces an estimate for C as its output. This estimate is fed into the Output Device Interface Module (D_OUT), which produces the output variables to drive the actuators.

The fourth step in the SCR process adds further design detail concerned with hardware failure and exceptions. A typical SCR specification will consist of monitored variables, a mode class and its modes, internal terms and controlled (output)

variables, conditions and events. The behaviour of these quantities are related by three types of table: Mode Transition Tables, Condition Tables and Event Tables. A condition is a predicate defined on a system state, whilst an event is a predicate defined on two system states that differ in the value of at least one variable [47]. The notation $@T(c)$ denotes an event which has the informal meaning "that condition c becomes true", whereas $@F(c)$ means "condition c becomes "false". Formally,

$$@T(c) \equiv \neg c \wedge c'$$

where the unprimed c is evaluated in the old (or current) state, and the primed c' is evaluated in the new (next) state. A conditioned event has the form $@T(c)\,WHENd$ which means condition c becomes true when condition d holds. Its formal meaning is defined by:

$$@T(c)\ WHEN\ d \equiv \neg c \wedge c' \wedge d.$$

A mode class is a state machine whose states are called modes and whose transitions are triggered by events [48]. The Mode Transition Table is a special form of the Event Table which defines the next state relation for the Mode Class.

A term is any function of input variables, modes, or other terms. They are useful for collecting together complex or repetitive behaviours and have a similar role to macros in RSML. Typically, a term will be used to make a specification more concise and its behaviour will be defined by an Event table. Often the controlled output variables can be defined as a condition on the current state, and are hence defined by a Condition Table. SCR has extensive tool support for editing, animation and model checking [46]. It includes tools to demonstrate well-formedness [45] such as: the Consistency Checker that demonstrates the specification is syntactically and type correct with no circular dependencies, no duplicate names or unused/unspecified variables, and no violations of disjointness or coverage properties; the Property Checker Salsa [10] further analyses the state machine; the Dependency Graph Browser graphically displays dependencies among the variables in the SCR tables; the Simulator for executing scenarios and evaluating behaviour; the Invariant Generator for checking that

state invariants hold. SCR has also been integrated/interfaced to model checking and theorem proving tools [45]. Further, SCR is still being developed and enhanced, e.g., currently SCR only supports primitive data types but this is being addressed by providing support for arrays and records [45, 46].

2.3.2 SpecTRM and RSML

SpecTRM [66, 70] is designed to assist in the development of software intensive safety critical systems. It forms part of a human centred, safety-driven process which is supported by an artefact called an intent specification [71]. An intent specification is stratified into a Program Management level (Level 0) and six further levels System Purpose, System Design Principles, Blackbox Behaviour, Design Representation, Physical Representation and System Operations [66]. The whole process is supported by a toolkit consisting of a specification editor, a simulator and analysis tools, and is designed in such a way as to support peer review. The formal model on which the tools operate corresponds to the Blackbox Level 3 of the intent specification, and is written in a language called SpecTRM-RL. A SpecTRM-RL model consists of four main components:

1. A specification of the supervisory modes of the controller being modelled

2. A specification of its control modes

3. A model of the controlled process

4. A specification of the inputs and outputs of the controller.

A major driver for the SpecTRM development has been the desire to remove implementation bias from the model. This is why a SpecTRM-RL model has a greatly simplified graphical representation compared with RSML (which is also based on the Parnas 4-Variable model), to remove internal behaviour which was found to be

problematic [73]. The mode concept has been introduced. A mode is just a collection of states, but its significance is to the user. It represents how a user or operator views the system being developed, it is not an internal implementation detail. For example, operating modes could be start-up, take-off, normal flight and landing for an avionics application. A major reason for including modes is to allow the modelling of the "mode confusion" concept, whereby an Operator action induces an error - perhaps leading to a hazardous situation - because he/she thinks the system is in a different mode of operation. A typical SpecTRM graphical model component has four quadrants, as shown in Figure 2.2.

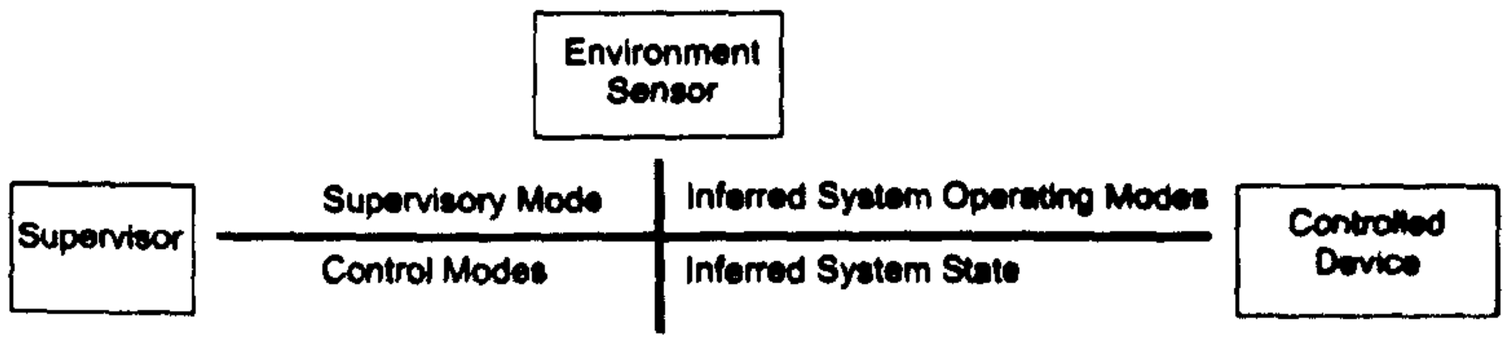

Figure 2.2: SpecTRM Graphical Model

The top-left is the supervisory interface, which consists of a model of the operator controls and the means by which status is communicated (displayed) to the operator. The bottom-left quadrant is the control modes of the controller, whilst the RHS represents inferred information about the operating modes and states of the system being controlled.

Interfacing to the component, are the Supervisor (control input, display output), the Controlled Device (Control command out, measured variable in) and the Environment Sensor (inputs measured variables). The graphical model is supported by a series of specification definitions which cover output message definition, input variable definition, and state variable definition. These definitions can be supported by further macro and function definitions which are aimed at improving the structure and readability of the SpecTRM-RL specification by abstracting complexity.

During trials of the RSML approach it was discovered that the use of propositional logic did not scale well to complex expressions in terms of its readability. It

was found to be error prone and that domain experts had difficulty in understanding what it meant [70]. The response was to develop a tabular representation of the complex logic, which presented it in disjunctive normal (DNF) form. These tabular representations were called AND/OR tables, and due to their success were retained for SpecTRM.

The SpecTRM-RL model represents the behaviour of the system as a state machine, and this allows a number of static checks to be applied to the model to verify its consistency (ensure the model is deterministic) and requirements completeness. The toolset supports a simulation capability, and the next phase in the analysis process is to run the simulator to allow the behaviour of the model to be validated. Once the analyst has confirmed consistency, completeness and behaviour of the model, then the final phase is to undertake the more heavyweight safety analysis. The toolset supports SMHA (State Machine Hazard Analysis) which is a backward search technique [104], and SDA (Software Deviation Analysis) [107] which evaluates the robustness of a specification.

2.3.3 Parnas 4-Variable Model Developments

Work at the University of York aimed to extend the Parnas 4-Variable model as part of the Practical Formal Specification (PFS) initiative [32] to address a number of development issues. The goal being to apply formality to embedded safety system development by providing two enhancements to the 4-Variable model. The first providing additional structure outside the control computer in the model, the second providing some additional structure within the computer.

The first, "outside", enhancement adds structure to the Parnas 4-Variable model to deal with two key issues. The first issue covers the concern about identifying more clearly the role of the embedding system (the control computer, its software and interfaces) and to ensure it is properly distinguished from its environment context.

This is achieved by enhancing the Parnas 4-Variable model by including embeddingInput and embeddingOutput variables which represent the inputs and outputs of the embedding system directly in the enhanced model. The second issue covers the concern that it is not always possible to monitor and/or control key properties directly through an environment parameter. The solution is to introduce additional variables into the model. These being real-world sensed and actuated variables which are affected directly by the system under development, and which are influenced by and/or influence the monitored/controlled variables in the original Parnas 4-Variable model.

The second, "inside", enhancement involves elaborating the 4-Variable model by expanding SOFTREQ (the computing system and software level specification) to include the hardware structure and the software architecture covering such issues as digitisation noise, sensor management and actuator dynamics data selection.

This work is based on developing the use of Matlab/Simulink/Stateflow (MSS) [18] by adding formalism to MSS specifications in a non-obtrusive manner (i.e. allowing those competent in using MSS to use formalism without the need for extensive training in formal techniques). This has three core elements which comprise: applying notational restrictions; representing important assumptions about the domain behaviour in MSS specifications; providing rules for "healthiness" of specifications which are checked by a Simulink/Stateflow analyser (SSA) tool.

The PFS/SSA approach addresses some of the identified issues and the authors acknowledge that it should be considered as one component of the work needed to address the major challenge of providing an integrated and usable approach for the development of safety critical systems.

2.3.4 KAOS

KAOS (Knowledge Acquisition in autOmated Specification) [24, 128] is an approach that provides a specification language for capturing why, who, and when aspects as well as the usual what requirements. It has a goal-driven elaboration method and draws on a number of concepts from Artificial Intelligence work on knowledge representation and acquisition [128]. A KAOS specification has a two level structure; an outer semantic net layer for declaring concepts, their attributes and links to other concepts; and an inner formal assertion layer for formally defining the concept. The latter combines a real-time temporal logic for the specification of goals, constraints, and objects, and standard pre/post conditions for the specification of actions and their strengthening to ensure the constraints [25]. KAOS is a form of goal-oriented RE that uses goals for requirements elicitation, elaboration, organisation, specification, analysis, negotiation, documentation and evolution. Goals are objectives to be achieved by the system under consideration. The term "system" refers to the software to be designed and its environment [68]. Goals are formulated in terms of optative statements which may refer to functional or non-functional properties and range from high-level concerns (the system will play audio warning messages to the pilot) through to lower level ones (if the message identifier does not correspond to the selected message then the audio mute discrete will be set active). A suitable goal directed acquisition strategy can be made up of the following steps:

1. Acquisition of goal structure and identification of objects.

2. Preliminary identification of agents and their actions.

3. Operationalisation of goals into constraints.

4. Refinement of objects and actions.

5. Derivation of strengthened conditions.

6. Identification of alternative agent responsibilities.

7. Actual assignment of actions to responsible agents.

Agents are active system components which act as a "processor" for some actions – an action being a mathematical input-output relation over objects. Achieving goals in general requires the cooperation of multiple agents. Although the steps are ordered, the strategy allows for some of them to run concurrently (e.g., 1, 2 and 3) and for backtracking and iteration. Tactics can then be applied to each step of the strategy, as detailed in [24]. For example the tactic "Reduce goals into subgoals so that the latter require co-operation of fewer potential agents to achieve them" can be applied to step 1 of the strategy. A typical model fragment after step 1 would have the form:

System Goal Achieve [MeetingRequestSatisfied]

Instance of SatisfactionGoal; **Concerns** .....; **ReducedTo** .....; **Informal Def** .....;

FormalDef

$(\forall r$: Initiator, m : meeting, p: Participant)
$Requesting(r,m) \wedge Feasible(m) \Rightarrow \Diamond_{\leq R(r,m)d}\ Scheduled(m)$
$\wedge\ Invited(p,m) \Rightarrow \Diamond_{\leq P(p,m)d}\ Know(p,m)$

The goal refinement process stops when goals are reached that can be assigned as responsibility to single agents. Terminal goals assigned to agents become requirements; terminal goals assigned to agents in the environment become assumptions. The later cannot be enforced by the software to be and have to be validated since the overall design will rely upon their properties.

KAOS supports separation of concerns between domain definitions (the environment) and system requirements and is consistent with the Parnas 4-variable model [19, 67, 133]. Further, the formal layer incorporates a library of reduction templates that are "pre-proved" using standard temporal logic techniques. Thus an analyst can use these templates with the knowledge that they are correct, and will not have to prove this later in the process. It is worth noting that the formal aspects of KAOS

were emphasised in the early work since it was found to be very useful in finding bugs in the resulting specification [128].

The use of KAOS in practice identified a number of issues concerning the very top-down nature of the approach. The solution developed was to introduce scenarios as discussed in detail in [129]. This effectively introduces a hierarchy of goals, requirements and scenarios.

2.3.5 AdLS

The AdLS approach is based on the belief that substantial improvement in the dependability of computer based safety-critical systems can be achieved by applying a detailed safety analysis to the requirements specifications of the software before proceeding to any subsequent phases of the development. The approach involves modelling the system and its environment, iterative simultaneous requirements and safety analyses, and documentation of linkages identified between artefacts produced by these analyses with a safety Specification Graph (SSG) [26].

The Methodology for the analysis of safety requirements involves the phases shown in Table 2.2, which provide a framework for partitioning the analysis into smaller phases obtained from a hierarchical model of the structure. AdLS uses a set of informal and formal techniques appropriate to the issues to be analysed. A typical AdLS development begins with an environment analysis to understand the environment context and to identify the accidents relevant to the system. This is followed by plant analysis that identifies rules of operation (assumptions) and plant safety specifications. The latter identify hazards that in conjunction with certain conditions in the environment could result in an accident. Safety constraints are then developed to counter the hazards and these are supported by plant safety strategies which are schemes for maintaining a safety constraint. The accidents, hazards, safety constraints and plant safety strategies can then be formed into a

hierarchical SSG. The SSG is used to direct the vulnerability analysis. The process proceeds with plant safety analysis followed by safety controlling system analysis.

Analysis	Purpose
Environment analysis	aim, purpose & accidents
Safety plant analysis	identifies hazards
Safety plant interface analysis	behaviour of sensors and actuators
Safety controlling system analysis	top level organisation for the controlling system

Table 2.2: AdLS Structure

At each phase the requirements analysis produces safety specifications, and the safety analysis checks acceptability - this is an iterative process. The safety specification imposes restrictions, and it is necessary to check these are consistent with the mission goals. Explicitly documenting assumptions enables impact of changes in them to be traced through the SSG.

2.3.6 Agenda-Based Methods

The goal of the agenda-based methods [44] was to counter some significant problems identified with the use of formal specifications: no methodological guidance given for the application of the formal technique; there is a need to perform requirements analysis before writing the specification; the formal techniques are not well integrated with the requirements analysis phase. The solution to these issues was to provide a requirements elicitation agenda to be followed which ensured that due notice was taken of analysing the requirements and distinguishing the environment, along the lines described in [133]. This agenda comprised of six items:

1. fix the domain vocabulary (domain theory, entities (nouns) and relationships (verbs))

2. state facts, assumptions and requirements in natural language

3. identify possible user operations and input/output parameters

4. list all relevant system events and their parameters

5. classify the events

6. formalise facts, assumptions and requirements as constraints on system traces

The starting point is expected to be an informal customer requirements document, and there is a need for communication with the customer to validate the process. The role of the first five steps is to understand the problem, the sixth is to initiate the formalism from which the formal specification can be derived. The latter also includes a heuristic approach for dealing with requirements conflict. The formal specification agenda is a two step process: define a first approximation of the software system state, and augment the specification by incorporating the requirements in turn. Completeness is identified as an important characteristic of the method, directly supported by the first steps of the requirements elicitation agenda and by feedback from the specification phase.

The agenda-based methods are not confined to any particular notation, but can be used with a number of formalisms. For example, later work demonstrated how Problem Frames and architectures could be used as part of a pattern-based specification and design approach following a four step agenda [17]. This involved providing additional problem frame templates to map to particular architecture styles.

The agenda-based approach has many similarities with KAOS. The main distinctions being that KAOS has its own language and takes a much broader perspective modelling the software system and the environment in detail. In contrast the agenda-based approach only models those aspects required to support an adequate specification. This means it is efficient, but raises concerns about how it might handle safety requirements which are system wide - i.e., includes the environment.

2.3.7 Formal Safety Analysis of Models and SCADE

Using a formal model to provide enhanced, formal safety analysis capabilities has emerged as an important research topic in recent years [58], [101], [2], [97]. Traditionally safety engineers perform manual analyses using the techniques described in section 2.2.2. These are based on informal design models and requirements documentation. The results tend to be quite subjective and dependent on the skill and experience of the analyst. There are also concerns that it is difficult to demonstrate that these manual analyses are complete, consistent and error-free [58]. This resulted in work which applied formal safety analysis to formal models of the system with the following advantages:

- close integration between the system and safety analyses as they use the same models
- the ability to perform a meaningful analysis of the architecture early in the development
- the ability to prove that the model has the required properties.

First the system formal model is developed and validated to establish nominal correctness, then failure injection is applied to investigate the system's safety capabilities. One approach identified a manual method for injecting failures [58], but other work noted the increased workload problem on large systems and developed tool support for automatic injection [101], [2]. A potential problem is that if the model is inadequate then so will be the analysis, therefore [97] encourages independence and diversity in the application of safety analysis techniques to ensure adequate coverage. In this work the POSE/Alloy [52] combination will be used to achieve a similar level of formal safety analysis of the formal requirements model and the PSA will be developed to support this task.

Both [58] and [2] used SCADE as the basis for their work, and SCADE was also used in recent work on developing AMBERS [23]. AMBERS also uses the Parnas 4-Variable model (based on the SCR variant) and tables for the requirements phase and targets the SCADE [31] system for the subsequent development. AMBERS has similar goals to and is also compatible with POSE, but it does not include the specific early (PSA) safety analysis feasibility check introduced in this work.

2.3.8 Automated Safety Analysis of Complex Systems

The importance of safety and embedded critical system development has been identified by a number of European Union (EU) funded research projects such as the ESACS (Enhanced Safety Assessment for Complex Systems), ISAAC (Improvement of Safety Activities on Aeronautical Complex systems) and ASSERT (Automated proof-based System and Software Engineering for Real-Time applications) [5] projects and through a number of pan-industrial collaborations using AADL (Architecture Analysis and Design Language) and the AltaRica project – AltaRica was used and further developed on the ESACS, ISAAC and ASSERT projects [11]. These projects have sought to improve the analysis and development of critical systems by addressing process issues and developing supporting analysis tools.

For example, an AltaRica model for use with PSSA [50] focuses on the potential fault propagation inside a system. Abstraction is used to reduce complexity in these models by replacing concrete variables (with potentially numerous values) that would be typically used in a functional model, with more abstract quality variables which have just two values – correct or erroneous. This abstraction produces smaller models, but at the expense of the AltaRica model not being the same as the functional development model - raising the obligation to demonstrate that the two model views are equivalent. The system hazards are represented as "feared events" which are encoded into the model using observer nodes. The analysis then proceeds

using tools to identify sets of unacceptable sequences (sequences where at least one feared event or qualitative system requirement is infringed). These unacceptable sequences are then analysed to identify the smallest set of software events that are required to eliminate these unacceptable sequences. This smallest set of software events is then formed into derived requirements which if satisfied ensure the unacceptable sequences cannot occur, and these derived requirements are added to the software specification.

One of the goals of the ASSERT project which ran from 2004 through to 2007 was to use AADL and proof techniques to improve co-operation between system and software development teams and produce re-usable building blocks to facilitate the development of complex embedded applications.

Automating and integrating the safety analysis tasks to cope with increasing complexity are features of these projects. Two prominent automated safety development approaches [77] used on these projects are those based on Failure Logic Modelling (FLM) and those based on Fault Injection (FI). FLM allows analysts to model the failure behaviour of a system as the design progresses and examples are AltaRica, FPTN and Hip-HOPS [77]. In contrast, FI allows automated analysis of detailed design models and uses SCADE and Simulink models as its input - examples being from the ESACS project and [58] (also refer to section 2.3.7). [77] argues that the two approaches are complementary as they deal with different parts of the development life cycle and they should be integrated - and this work was developed as part of the ISAAC project. [77] also notes that even FLM represents a significant cost and suggests the need for "lightweight" methods to be used in the early phases of a development to assist with such things as architecture evaluation and selection - and notes that [76] is an example of a possible lightweight PSSA approach.

The lightweight PSSA approach described in [76] is based on software architectures (although with some modification to the process it can be extended to cover hardware and system architectures) and consists of three main phases: dependency

modelling with both forward and backward model views; deductive exploratory analysis; inductive verification analysis.

In dependency modelling two views (called Forward and Backward Dependency Views) of the architecture are constructed to capture information about dependencies between services of different modules of the architecture – where a service could be a delivery of a data item, process communication or synchronisation. The two dependency views are then used in the analysis phases of the assessment.

The task of the deductive exploratory analysis is to identify and analyse potentially hazardous chains of dependencies in the architecture. These chains are identified by considering hazards (identified at earlier stages of the safety process), specifying them in terms of the services that modules provide, and then walking through the views in order to establish all the dependencies of critical services, as well as any mechanisms that minimise these dependencies.

Finally, in the inductive verification analysis hypothetical failures of some key modules services are considered and their potential effects on the system as a whole are established. Therefore, the purpose of this phase is to verify both the completeness (in terms of hazardous dependency chains identified) and correctness (especially with respect to potential single causes of failure) of the deductive analysis.

Each of these three main phases is broken down into a number of sub-phases which breaks the overall analysis down into manageable incremental tasks and provides verification for important aspects of these tasks. It will be shown that the development of the POSE/Alloy [52] combination in this work could also form a possible lightweight PSSA approach which could be used at the front end of the overall process described in [77].

2.4 Problem Orientation

By problem orientation (PO) we mean a body of work whose origins can be traced back to the seminal "Four Dark Corners of Requirements Engineering" paper by Zave and Jackson [133]. This paper identified four weak or *dark* areas in requirements engineering (RE) and concluded that there is a need in RE:

1. for terminology to be grounded in the reality of the environment;

2. to avoid describing the solution machine, rather describe the environment before and after the machine is applied;

3. to identify which entities are controlled by the environment, which are controlled by the machine and which environment entities are shared with the machine;

4. to recognise that the primary role of domain knowledge is in supporting the refinement of requirements into specifications.

The PO approaches find their genesis in these four issues:

1. for terminology to be grounded in the reality of the environment; Problem: the reality of the environment has no precise description so that requirements exist in the real world which consists of some inherently non-formal and hence imprecise elements. The difficulty for software is that software concerns the operation of the machine, a formally describable system, so that software problems (i.e. real-world problems that have software as their solution) consist of partly imprecise and partly precise elements [53]. This complicates the requirements sub-life cycle, identified by Lamsweerde [127] as consisting of Domain analysis, Elicitation, Negotiation and agreement, Specification, Specification analysis, Documentation and Evolution. With major real-world concerns such as sociological, political, organisational and economic issues being apparent to

the requirements, as well as formally describable issues of computation such as control flow, parallelism, etc. the path from requirements to code is required to combine both informality and formality. PO approaches, with their focus on the problem, must attempt to address the informal of the real world and the formal of the machine together. Also, any problem oriented approaches must be sufficiently expressive. This requires two capabilities [133]. The first is that they must provide for the declaration of a finite collection of action types that partition the space of all possible actions. The action types are (a) unshared environment controlled (UEC), (b) shared environment controlled (SEC) and (c) shared machine controlled (SMC). The second expressive capability is that the languages must allow assertions about the three action types, in both the indicative (about the environment) and optative (about the requirement) moods.

2. the source of this dark corner is the mad rush to a solution that overcomes the need to understand a problem before it is solved, avoiding the description of the solution machine in the requirements forces a consideration of the complexity of the context and requirements of the problem. However, it is easy to state that you should concentrate on understanding the problem before undertaking the solution, but this can be very difficult to achieve in practice. As a result, PO approaches tend to provide tools for the manipulation of the problem, rather than the of the solution. The problem world is a big place and it is not always straightforward to identify the relevant parts of the environment. In such cases domain knowledge and experience are essential to success, as is appropriate modelling and analysis [127]. There is a need to include just the right parts of the environment. Too much information involves wasted effort and potential obfuscation by irrelevant detail, too little and essential behaviour may be omitted. The former is wasteful, but the latter is potentially

disastrous for the development since the developed machine will not interact correctly with its environment. Systems theory introduces the concept of "emergence", which is the idea that at a given level of complexity there are properties characteristic of that level [16]. It amounts to the idea that the "sum" is somehow more than the "parts" and presents a problem for the related concept of reductionism, so important in many aspects of science and which has a long history [78, 115]. The reductionist approach to complexity is to partition the problem into simpler tasks and attack these separately, and this has proven to be of immeasurable value in many areas of science, particularly the physical sciences of chemistry and physics [16]. However, the concept of emergence means that checks have to be introduced to ensure that something important is not lost through the application of reductionism. Some problem oriented approaches split the problem into simpler parts and work on these: only considering the composition problem of combining these parts after the solutions to the individual parts has been successfully established. This "divide and conquer" approach follows from the premise "if you cannot solve the smaller problems, then you will not be able to solve their composite" [55]. It recognises that the composition may be difficult, but argues this is facilitated by the knowledge gained from solving the simpler problems and not having to try to solve the simpler problems at the same time as dealing with the composition.

3. the separation of environment and solution is another aid in managing the complexity of the problem as it allows the specification to be distinguished from the requirements. These are both optative entities, but the requirements are formulated in terms of objects in the real world in a vocabulary accessible to the stakeholders. Requirements capture required relations between objects in the environment that are monitored and controlled by the software [19]. In

contrast, the specification is formulated in terms of objects manipulated by the software in a vocabulary accessible to programmers and captures required relations between input and output software objects. There is a semantic gap between the requirements and the specification which has to be bridged for a successful development to take place, and domain knowledge of the environment has a key role in bridging this gap. An important aspect of the PO approaches is the tools they provide to assist this bridging process. In [133], Jackson and Zave noted that requirements that are not implementable fail to be so for three general reasons:

(a) Environment Constraint. Some requirements can only be satisfied by constraining an action that is controlled by the environment. For example, in a turnstile entry system there might be a requirement that the number of people allowed entry must not exceed the number of payments received. This represents a constraint on the entry actions which are controlled by the environment.

(b) Unshared Information. Some requirements are not directly implementable because they are stated in terms of unshared phenomena. For example, a lift must arrive at a floor where people are waiting - how can the machine know that people are waiting? The resolution of this problem requires request and arrive actions to be shared with the machine. It also relies on the domain knowledge about human behaviour that people will tend to wait after they have made a lift request.

(c) Future Reference. Some requirements are not directly implementable because they are stated in terms of the future. A high-level requirement from an aircraft stores management system might state "A store on a store carrier selected for release, must be successfully released before another store from the same carrier can be selected for release. If the first

store is not successfully released, then it must be declared as *hung* and no further store release attempts are allowed from that carrier". The stores management computer (SM) does not know if the store has gone or not. The SM provides the control signal to set the release process in motion, but it has no way of knowing (by itself) whether the release - which is a future event - is successful or not. To satisfy the requirement, the SM has to take in data from the environment. In this case data from the store carrier stating if the store has gone or not, and knowledge that the explosive devices used in the release mechanism will have completed well within a 500 millisecond limit. This knowledge allows the requirement to be re-expressed in a more implementable form as "A store on a store carrier selected for release, must be successfully released within 500 ms, otherwise the store will be declared as *hung* and no further release attempts to the carrier should be allowed. Successful release is defined as the store carrier informing the SM that the store has gone". This implementable form of the requirement includes domain knowledge and relies on environment controlled phenomena.

Note that all of these types use domain knowledge to *fill-in* behaviour information to make them implementable. In fact these three types are not "non-implementable" in the strictest sense, but rather they cannot be implemented directly without the addition of further information. Therefore this text will refer to them as being "non-feasible" rather than "non-implementable" – the idea being that they can be implemented if the required information is supplied. In fact, Zave and Jackson [133] classified requirements into four types, the above three non-feasible requirements and they also identified a fourth class that are directly implementable.

4. formally Zave and Jackson express this in the entailment:

$$W, S \vdash R$$

In the entailment W represents the indicative properties of the environment domain (the world context), S is the specification of the machine and R represents the desired or optative properties of the requirement. The entailment is taken to mean that if a machine satisfying the specification S is operating in the world with environment domain (indicative) properties W then the optative properties defined by the requirement R will be satisfied. That is, the relationship between the software solution and the requirements is given by the context into which the solution is deployed. Satisfying this entailment is an important first task in the software development process, and involves understanding the environment domains and establishing the requirements that will result in the desired behaviour before specifying the computer machine that forms the solution.

The completeness of requirements engineering can be formally formulated in terms of this entailment:

(a) There is a set R of requirements. Each member of R has been validated (checked informally) as acceptable to the customer, and R as a whole has been validated as expressing all the customer's desires with respect to the software development project.

(b) There is a set W of statements of domain knowledge. Each member of W has been validated (checked informally) as true of the environment.

(c) There is a set S of specifications. The members of S do not constrain the environment, they are not stated in terms of any unshared actions or state components, and they do not refer to the future.

(d) A proof shows that $W, S \vdash R$. This proof ensures that an implementation

satisfies the requirements.

(e) There is a proof that S and W are consistent. This ensures that the specification is internally consistent and consistent with the environment.

In the following we will review a number of approaches which have developed from these ideas.

2.4.1 Problem Frames

Problem Frames (PF) [53] allows one to construct the specification (S) for a machine (M), such that the machine M satisfying S, in conjunction with properties of the problem world, W, satisfies the requirements represented by R. This covers part of the $W, S \vdash R$ entailment described in section 2.4 and is represented diagrammatically in Figure 2.3.

Figure 2.3: A Typical PF Problem Diagram

The machine M and the problem world W are both physical entities, so they are shown as solid lines in Figure 2.3. In contrast, the requirements R are intangible so are shown dashed. The machine interacts with the problem world at an interface represented by the shared phenomena **a**. Shared phenomena are either controlled by one of the machine or the problem world and observed by the other [55]. The requirements refer to the physical phenomena, represented by **b**, of the problem world. The arrow denotes that the requirements place constraints on these phenomena. In general, the requirement phenomena **b** are distinct from the specification **a**. Although they can be the same and it is often convenient to achieve this, if possible, as it simplifies the analysis.

PF are targeted at real world engineering problems, but not to what might

be called *computational problems* such as factorising a large integer or finding the shortest path in a graph [55]. The latter are ideal candidates for formal development [43] due to their closed nature (i.e., no interface to the real world). PF can also work with formal development systems [113, 114, 17], but as noted in section 1.1 care is required in mapping between the real world entities and their formal equivalents.

The gap between what is desired at the interface **b** (requirement R) and what can be directly monitored and controlled by the machine at the interface **a** is bridged by the given properties of the problem world W as shown by $W, S \vdash R$ and explained in section 2.4. Showing this relationship holds is called the *basic problem concern*. Therefore, a PF development begins with a description in the real world involving R, **b**, W, S, and **a**, and from this a set of related sub-problems are derived – where the sub-problems map to known patterns called basic frames. Jackson identified five basic frames [53] these being:

- *Required Behaviour*: the problem is to build a machine that will control some part of the physical world to satisfy certain conditions.

- *Commanded Behaviour*: the problem is to build a machine that will control some part of the physical world in response to commands issued by an operator.

- *Information Display*: the problem is to build a machine that will obtain information about the physical world and display it appropriately in the required form.

- *Simple Workpieces*: the problem is to build a machine that will allow a user to create and edit certain computer text, graphics or similar objects so that they can be processed as required.

- *Transformation*: the problem is to build a machine that will transform given input files into output files in the required format.

PF are the most researched of the problem oriented approaches and have an active research community developing and extending their scope [20, 21]. Some authors have provided extensions to the basic set of PF templates to cover a particular problem domain. PF have also been used in conjunction with architectural styles to form a systematic software development process based on patterns and the agenda concept [17]. Patterns are a way to reuse software development knowledge at different levels of abstraction. In this scheme PF provides problem patterns and the architectural styles provides solution patterns. The agenda encodes process knowledge. It is the combination of the patterns and the agenda that provides the systematic development process.

A more generic extension to the PF approach is the architectural frame or AFrame, which provides an architecture-driven problem decomposition [103]. This uses AFrames as architectural styles located in the solution space to guide the analysis of the problem space. AFrames have been developed for the linear pipe and filter, and model view controller (MVC) architectures to show the utility and power of the approach. They also address two criticisms raised against PF: the assumption of a green-field development process and expertise is required to guide the appropriate decomposition [103]. The AFrame concept addresses these criticisms as follows. AFrames provide guidance on how the problem needs to be restructured to fit a particular solution form. They also assist in the solution synthesis by guiding the decomposition of sub-problem solutions into a solution of the original problem. The AFrame structure encodes expert knowledge, which directly supports the software development process. In this way, AFrames provide a link between the problem and solution spaces.

In their standard form [53] PF are focussed on the problem domain and do not have tools to bridge the gap and access the solution space. The development of AFrames addresses certain aspects of this issue, while POSE, which is considered in section 2.4.3 below, takes the work even further.

2.4.2 REVEAL

REVEAL [42] provides a process to elicit the real world requirements in the context of the proposed operating environment (domain). It provides a means to develop the specification of a machine that will satisfy the requirements when operating in the context of the environment it is to be used in. The process is supported by checklists to ensure adequacy, quality of expression and self-consistency of the requirements. The process also provides support for managing and resolving requirements conflicts and for managing change. Although REVEAL is a process it is tool independent. It is informal in operation, but can be used as part of a formal development. It can also be used in conjunction with other common requirements tools such as DOORS.

One of the key principles of REVEAL is that Requirements Engineering begins with a description of the whole application domain, including domain entities that do not directly impact with the system. Iteration covers the transformation from high level requirements about the world into specifications about the machine and its interface. Scenarios are used as a tool for describing and exploring such dynamic behaviour. This approach readily establishes traceability, but goes further with its notion of *Rich Traceability* – which provides heuristics to support the development from the requirements into the specification. For a given requirement, its *Rich Traceability* consists of an argument that explains:

- which statements (e.g., requirements, specifications or domain knowledge) combine to satisfy it;

- how/why the particular combination achieves the given statement.

Therefore *Rich Traceability* plays the role of the turnstile ($\vdash$) in the satisfaction argument.

2.4.3 POSE

Problem Oriented Software Engineering (POSE) [39, 40, 41] is an extension and generalisation of Jackson's Problem Frame approach [53]. The following is a brief summary of its main characteristics based on the more detailed descriptions in [39, 40, 41]. In POSE, software development is viewed as solving a problem, the solution being a machine (a program running in a computer) that will ensure satisfaction of the requirement in the given problem world consisting of real-world domains. Like Problem Frames, POSE views the problem world as a collection of domains described in terms of their known, or indicative, properties, which interact through their sharing of phenomena, i.e. events, commands, states, and so on.

POSE is defined as a transformational system, similar in form to a Gentzen-style sequent calculus [63] that allows problems to be transformed into problems that are easier to solve, or that will lead to problems that are easier to solve. A set of transformation rule schema capture (atomic) discrete steps in development. Each requires a justification of application in order for the transformation to be solution preserving (this means that a solution to a transformed problem is also a solution to the original problem), but the justifications need not be formal. The combination of the justifications is an argument that the solution is adequate as a solution to the original problem.

POSE provides a structure which supports different development activities and allows them to be combined – this includes bringing together both non-formal and formal development activities. Development commences with the instantiation of the null problem about which little is known, and software development then proceeds by the application of problem transformations, each of which develops some aspect of a problem description – including gaining knowledge about the requirement and the environment context. The choice of which transformation to apply at any point is only tentative as each transformation must be justified before being

accepted as contributing adequately to the development. Sometimes a sequence of transformations will lead the problem to a "dead-end" and it will be necessary to backtrack to the point of departure and then follow a different path to an adequate solution. Note that these "dead-ends" provide useful insight into the problem space. This information can be very useful to help avoid cases where for example an over-enthusiastic maintainer may want to follow a path that has previously been shown to be a "dead-end". POSE is the basis of our candidate SSSD approach and will be introduced in detail in Chapter 4.

2.5 Literature Analysis

In this section we analyse the SSSD approaches described in section 2.3 with a view to identify some key properties that they exhibit. We then use such properties to evaluate the PO approaches described in section 2.4 with a view to justify a candidate approach to use in the remainder of this thesis.

2.5.1 Property Identification

Inspection of the SSSD approaches described in section 2.3 allows a number of key properties to be identified that an approach targeted at the development of the early phases of a safety development process should possess. The following paragraphs identify ten such properties.

Understanding the system environment context (**Context**) is a central feature of the Parnas 4-Variable based techniques such as Parnas Tables, SCR, SpecTRM, AMBERS and the PFS development work (section 2.3.3). It is also important in the AdLS, FLM and KAOS approaches, although in KAOS it is not given the same level of prominence as other features.

One way of reducing the risk of rework is by selecting an appropriate system architecture that facilitates the resolution of the key issues as an integral part of the

development. Without such a selection, then one may end up with an inappropriate architecture that is not capable of assisting with the resolution of all or even some of the key issues. The result is that the development of a satisfactory solution becomes much more problematic and inevitably more expensive. So the capability of modelling different candidate architectures is a highly desirable feature of any proposed approach (**Architecture**). KAOS, PFS and FLM have features that provide strong support for architecture, whilst the other techniques (Parnas Tables, AdLS, SCR, SpecTRM, and AMBERS) provide some support.

All of the safety techniques discussed in section 2.3, apart from AdLS, have some capabilities in producing a specification that spans the gap between the problem world and the solution space. The full process approaches of KAOS, AMBERS, SCR, SpecTRM and Parnas Tables cover this in some detail, but it is not a specific highlighted feature of PFS or the FLM approaches. It follows that the language of the selected approach must be able to span the problem and solution spaces (**Spec. Join**).

Implementation bias (**Avoid Bias**) can result in unnecessary and inappropriate constraints on the solution, and the avoidance of this property is a desirable. The KAOS approach has good capabilities in dealing with this property through its multi-level abstraction and validity checking mechanisms, and the linkage and focus provided by the SSG means AdLS is also good at avoiding bias. The high level of abstraction and other features also mean that the FLM approaches and PFS have good capabilities in avoiding bias – but the other techniques are not so strong in this area.

The Parnas 4-Variable based techniques ensure that the model, domains and interfaces are grounded in reality (**Model Reality**), and this property is also an important feature of PFS with its enhanced features. The SSG linkages mean AdLS has good capabilities with respect to this property. However, it is not a strong feature in KAOS or the FLM approaches.

All the techniques support the task of validating that the right system is being produced to a certain extent (**Validation**). Some, like SpecTRM, SCR, PFS, the FLM approaches, KAOS and AMBERS go further by providing tool support for validation, including simulation and/or proof.

KAOS manages complexity through its ability to structure the problem, so that large complex problems can be broken down into smaller problems that are easier to solve (**Simplification**). The other approaches such as AMBERS, Parnas Tables, SCR and SpecTRM use simplification as part of their development process, but do not have the process support offered by KAOS. PFS offers strong capabilities with its modelling and refinement support, whilst the FLM approaches make use of their "divide and conquer" features such as abstraction to handle complexity.

Tool support (**Tool Support**) is an important practical consideration with respect to both efficiency, error-avoidance and repeatability, so this is certainly a desirable property and techniques such as the FLM approaches, PFS, KAOS, SCR, SpecTRM and AMBERS support this. Specific tool support is not a feature of Parnas Tables or AdLS.

Safety standards require analysis to be performed early in the life cycle, so the ability to work at a suitably early point, i.e., the right level of abstraction, is also important (**Right Abstraction**). KAOS, PFS, AdLS and the FLM approaches have very good abstraction capabilities and support comprehensive analysis tasks, as does AMBERS. SCR and SpecTRM also support extensive analysis capabilities although this tends to be relatively later in the life cycle. Parnas Tables do not have analysis tool support but allow a manual analysis to be conducted.

Finally, one of the aims of this work is to improve an existing process (i.e. to support "normal design" principles), so the selected approach must be capable of working with an existing life cycle method (not replacing it!) – i.e. it must integrate with it (**Integrates**). Approaches such as KAOS, PFS, SCR, SpecTRM, and AMBERS have a specific tool supported process which is different from the existing

process introduced in detail in Chapter 3. These approaches could be adapted, but it would be a significant undertaking. In contrast AdLS and Parnas Tables could be adapted for use with the existing process and the FLM approaches could be used directly.

Although these ten properties were derived from an analysis of the SSSD approaches presented in section 2.3 they are also supported by other sources in the literature and all this is summarised in Table 2.3.

Property	Description	Literature Support
Context	Importance of Domain Context	[54, 82, 121, 127]
Architecture	Architecture to encapsulate	[82]
Spec. Join	Specification as "join"	[121]
Avoid Bias	Avoid implementation (solution) bias	[133, 16]
Model Reality	Model domain and interfaces grounded in reality	[133, 16]
Validation	Validate the model represents reality	[122, 16]
Simplification	Support simplification	[69, 16]
Tool Support	Tools to support application of approach	[96, 45, 90]
Right Abstraction	Early analysis requires abstraction	[82, 71]
Integrates	Integration with Existing Life Cycle	

Table 2.3: Ten Properties of SSSD Approaches

A comparison of the relevant SSSD approaches with these ten properties is shown in Table 2.4, where the property descriptor from Table 2.3 is used to represent the desired property as appropriate. The FI techniques (section 2.3.8) apply to detailed design models so are not considered for the early development phase work that is the focus of this document. In this comparison table (Table 2.4) "Y" denotes that the approach supports the property based on supporting evidence from the literature, "P" means the approach partially supports the property or might support the property but there is insufficient evidence from the literature, whilst "N" means the approach provides little or no support for the property based on the published literature.

The Parnas Table based methods are strong contenders, since they were developed to produce safety systems, and the system context and handling requirements are an important aspect of their operation, but they are more appropriate to the

Property	SCR	SpecTRM	Parnas	KAOS	AMBERS	AdLS	PFS	FLM
Context	Y	Y	Y	P	Y	Y	Y	Y
Architecture	P	P	P	Y	P	P	Y	Y
Spec. Join	Y	Y	Y	Y	Y	N	P	P
Avoid Bias	P	P	P	Y	P	Y	Y	Y
Model Reality	Y	Y	Y	P	Y	Y	Y	P
Validation	Y	Y	P	Y	Y	P	Y	Y
Simplification	P	P	P	Y	P	Y	Y	Y
Tool Support	Y	Y	N	Y	Y	N	Y	Y
Right Abstraction	P	P	P	Y	Y	Y	Y	Y
Integrates	N	N	P	N	N	P	N	Y

Table 2.4: Comparison of Properties by SSSD Approaches

design rather than the early development phases. SCR, SpecTRM and AMBERS have extensive tool support, but the modelling supported by these methods is again more suited to the design phase. Neither are they very strong on the architectural aspects. They provide some support for simplification and can avoid bias, but this is not a specific feature of these approaches. Finally, they offer little direct support for the integrates property. simplification.

AdLS has many good features but lacks tool support for the formal techniques associated with it which affects its validation capabilities, and does not have strong support for the Architecture or Spec. Join properties. However, with some work it could be integrated with different development approaches.

In many respects KAOS is a strong contender, it has good tool support, supports a combination of formal and informal development and has strong elicitation capabilities. However, although KAOS does note the importance of the environment and that it is necessary to distinguish between the requirements and the environment; its environment context capabilities are not strong when compared with the PO approaches, and as a result the KAOS models are not as "grounded in reality" as desired. KAOS is also weak with respect to the integrate property due to its prescriptive nature.

That PFS features strongly is no surprise since it was conceived to handle em-

bedded safety critical applications and took the Parnas 4-Variable model approach as its basis, but added enhancements to the model and formality – "where formality adds engineering value" [32]. The Spec. Join property is not (currently) specifically supported and it is centred on MSS (Matlab/Simulink/Stateflow) – therefore it will not integrate easily with other development approaches.

Finally the FLM approaches have good capabilities and AltaRica was developed for avionics applications. However, they are primarily approaches for achieving automated safety analysis of complex systems (hence Model Reality and Spec. Join are specifically supported) and need to be used in conjunction with an appropriate development approach.

2.5.2 Assessment of the PO Approaches

In this section we assess the problem oriented approaches of section 2.4 against the identified properties:

- **Context**: the importance of context knowledge is one of the basic tenets of problem orientation, hence characteristics of all PO approaches reviewed.

- **Architecture**: This is not part of the original work of [133]. The architecture work of [103] partially addresses architectural design within problem frames. POSE fully supports architectural design as part of its basic transformation (as we will see in Chapter 4).

- **Spec. Join**: of the reviewed approaches, POSE has spanning problem and solution spaces as one of its key objectives.

- **Avoid Bias**: avoiding solution bias is a characteristic of the PO approaches, which stress the importance of fully understanding the problem, before designing a solution, and of expressing requirements solely in terms of the context phenomena.

- **Model Reality**: the concept of designating descriptions and phenomena in the real world is key to many PO approaches, particularly Problem Frames and POSE.

- **Validation**: a notion of validation is present in Problem Frames embodied in the concept of basic problem concern. It is a central concept in POSE where the notion of adequacy with respect to stake-holders and step justification come together to provide explicit tools for validation.

- **Simplification**: the idea of problem progression was introduced within Problem Frames to signify the move from requirements to specification during requirements analysis. It was then developed, independently into techniques by [74, 75, 113, 114]. POSE defines its own transformation notation of problem progression.

- **Tool Support**: none of the PO approaches offer any specific automatic tool support, except perhaps for a prototype Prolog-engine for POSE transformations [38]. From the literature there is no evidence of integration with other tools.

- **Right Abstraction**: all PO approaches are permissive when it comes to the choice of descriptions, so that identifying an appropriate level of abstraction remains an open problem for all of them. However, some guidance might be derived from Zave and Jackson's requirements taxonomy and their characterisation of implementable specification [133].

- **Integrates**: the only PO approach which so far has been integrated into practice is REVEAL, which in fact was defined in an industrial context.

From this analysis it follows that PO approaches in general, and POSE in particular, offer good capabilities with respect to the ten properties identified in Table 2.3.

By definition all PO approaches focus on gaining an adequate understanding of the system context and the requirements. Problem Frames scores well on most of the criteria, except that it is weak in addressing the solution domain and has no tool support. In contrast, POSE is capable of scoring highly on all the criteria except for tool support. Therefore, on balance POSE appears to offer the best fit with respect to the properties identified in Table 2.3. However, evidence will need to be collected to support this proposition, and this will be covered in the chapters that follow.

2.6 Chapter Summary

This chapter began with a definition of the safety terminology to be used in this thesis, followed by an investigation of the safety context that underpins this work – including the need to produce evidence to support a safety case. The need to perform a safety analysis in the early phases of the development life cycle was justified, and some techniques that might be used to achieve this were introduced. Then a number of safe software system development (SSSD) approaches were presented. Next problem orientation was introduced, the concepts involved explored and some problem oriented approaches discussed. The chapter concluded with the identification of the ten generic properties based on the SSSD approaches, against which the SSSD and the problem oriented approaches were compared. The comparison indicated that Problem Oriented Software Engineering (POSE) appeared to be capable of addressing most aspects of the ten properties and was therefore selected as the candidate approach to be used in the rest of this thesis.

Although much of this work will concentrate on the early phases of the development process and the task of deriving a safe specification from the system requirements, it is important to note that the selected method must have capabilities that can span the gap between the problem and solution spaces, because the ultimate aim is to develop safe code to implement the safe specification. POSE has these

capabilities, providing further evidence to support its selection for this work.

Chapter 3

Analysis of an Industrial Safety Process

Chapter 2 identified a set of ten key properties that an appropriate safe software system development (SSSD) approach should possess. The aim of this chapter is to describe and analyse the early phases of an industrial safety development process with a view to identifying additional properties and process issues the discharge of which could lead to a notable improvement of the process, and that our proposed approach should address.

3.1 Process Overview

The specific aspects of the safety development processes followed by different companies depends on the standards they are required to follow and this is often determined by the business sectors they operate in. The author's company (the Company henceforth) operates primarily in the UK defence sector, so its safety process is based around DS 00-56 [125]. As will be explained there are some unique aspects of the process, but generally it is typical of the safety processes used by other UK companies operating in the military sector, and it shares many similarities with safety

processes in general (e.g those following IEC 61508 [51]).

The Company follows an integrated product development process (IPDP) for all their development projects, including safety – thus satisfying the goal discussed in section 2.2 that the safety process should be integrated with the development process. For developments with functional safety requirements the basic process is augmented with high integrity activities; these will be described where appropriate in the description of the IPDP that follows. The IPDP is split, as shown in Figure 3.1, into an over-arching Management phase and three main technical phases which are historically called: (a) Concept Evaluation (CE), (b) Demonstration/Validation (Dem/Val) and (c) Engineering, Manufacturing and Development (EMD). With the latter phase being by far the major component, it is further sub-divided into the major sub-phases of Preliminary Design, Detailed Design, System Validation and Certification. The IPDP has been used for over 10 years and has evolved as a result of feedback from its use on a wide variety of projects. This evolution has meant that some of the phase/task titles no longer capture completely all the activities undertaken to support them, as will become clear when some of the tasks are described in more detail below. It should be noted that Figure 3.1 concentrates on the initial part of the IPDP since this is the area of interest for this work - namely the CE phase, the Dem/Val phase and the EMD Preliminary Design sub-phase. These phases are further sub-divided into high-level tasks, which in turn are further divided into sub-tasks and so on, to form a hierarchical task structure. That is, the development process is partitioned into a series of inter-related tasks. These tasks cover management, planning, analysis, development, validation, verification and review activities; and as noted, the safety tasks are an integral part of the structure. Each task details the input criteria, what is to be achieved by executing the task and the output criteria. Following this IPDP has given the Company a repeatable, consistent and controlled development process.

The exit from one phase/sub-phase to entry to the next is controlled by a major

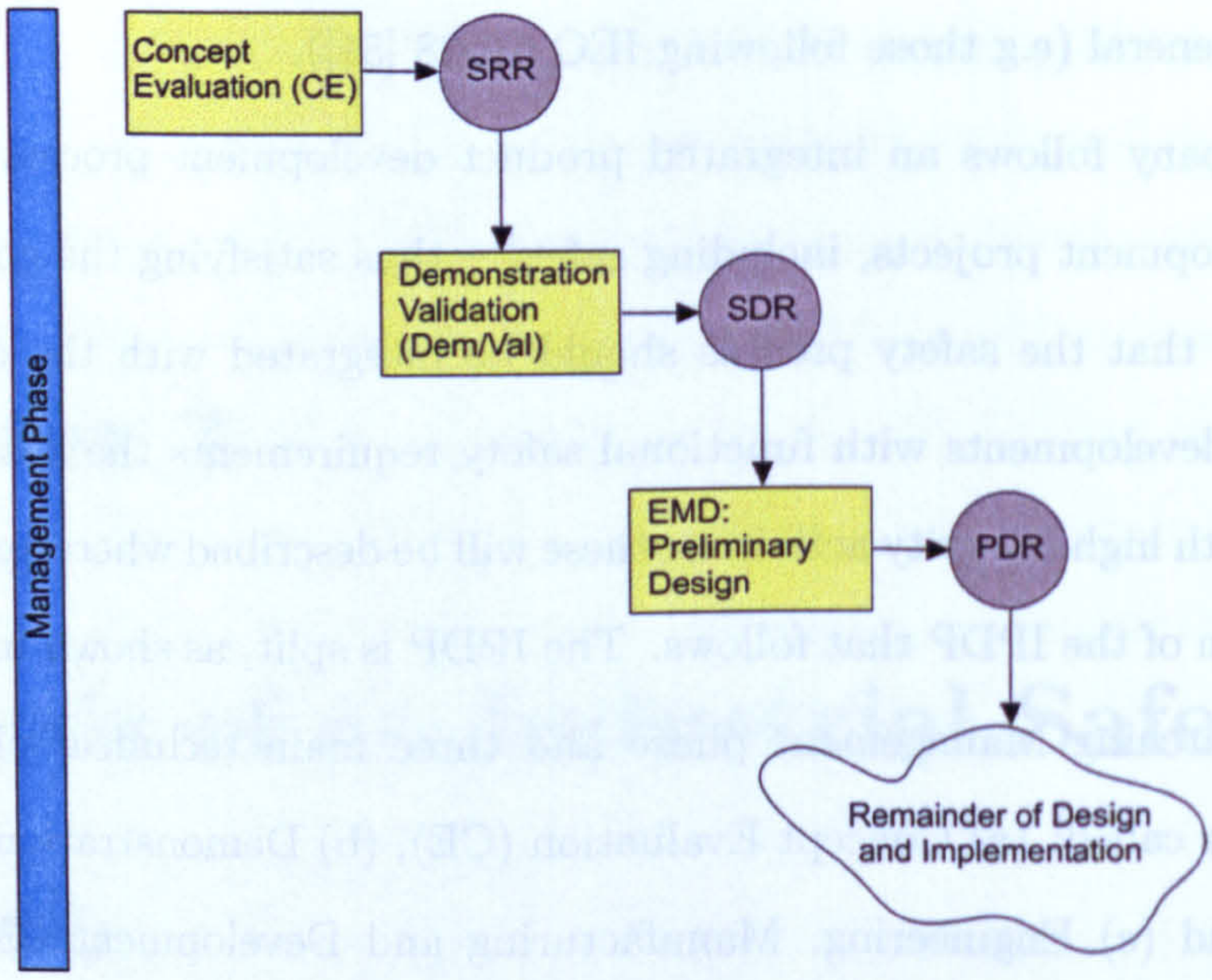

Figure 3.1: The Integrated Product Development Process (IPDP)

review - these are the SRR, SDR and PDR on Figure 3.1 and explained in more detail below. The transition between phases is only allowed if appropriate review criteria are satisfied.

The safety management tasks of safety planning (output being the Safety Programme Plan) and safety audit and monitoring are included as part of the Management phase. The IPDP safety tasks are based on a safety life cycle model that follows the approach defined in the second edition of DS 00-56 [123]. These IPDP safety tasks are under continuous development to ensure that they satisfy the safety goals of the products being developed. The later version of DS 00-56 [125] is goal oriented and does not prescribe what sort of life cycle should be followed. However, it does identify the type of safety artefacts that should be produced (including hazard list, hazard analysis and risk mitigation) and requires that a safety case should be developed to provide the argument for safety. A detailed review [84] shows that the IPDP safety tasks provide the required safety artefacts and a safety development process that supports the production of an appropriate safety case argument. Thus the IPDP safety process has been evolved such that it continues to satisfy the

provisions of DS 00-56 as the latter is developed over time.

All projects in the Company have to follow the safety life cycle built into IPDP, since every project has safety obligations that must be discharged. For many projects, these are just non-functional health and safety based obligations which discharge the Company's obligations to satisfy EU Directives and their corresponding UK safety regulations. However, for safety involved and safety critical developments the level of analysis and need to provide sufficient evidence increases many fold, and the safety case is a much more detailed structure. It is the latter types of system that this work is interested in supporting. Experience within the Company has demonstrated that these safety systems require much more work to cover their functional behaviour aspects. This is characterised by the need for:

- development of a logical argument for safety
- specific architectural features to support the safety argument
- significantly more and diverse evidence, including proof for safety critical work.

Using this process to build these architectures and adopting a policy of "keep it simple" has resulted in systems that have been accepted into service at the highest levels of criticality by the various regulatory authorities. The Company is keen to continue along this successful path, but is also keen to continue to improve the integrity and efficiency of the IPDP safety process. To support this the Company runs a *Lessons Learned Database* (LLD) which is used to record the good and the not so good elements of all completed programmes. The aim is to capture any problem issues arising so that (a) the lesson is recorded and is learned, and (b) corrective action is implemented to avoid the same problem in later programmes. Discussions on and comparisons with the safety development processes used by other companies operating in the sector indicate that these processes share the same issues and problems [1].

[1]The Company works with other industry representatives through its presence on a number of

In the following sections, we describe enough of the early phases of the IPDP to allow the identification of a necessary property and some advantageous properties that we would like our development approach to possess. In addition, we identify six issues, instances which are repeatedly found in the LLD that will be addressed in our work. To this end, we will separate and provide a short description of each issue that will allow them to be traced through the remainder of this work. Although most of the issues are observed in the CE phase, they are also issues with the following phases; after their description and capture from a description of the CE phase, we continue to identify their impact on the subsequent phases in section 3.3 and section 3.4.

The case study work in Appendix A shows an example of the tasks involved in the CE, Dem/Val and EMD Preliminary Design phases – this section can be read without reference to the Appendix, but the latter does provide additional information if required.

3.2 Concept Evaluation (CE) Phase

The first part of the CE phase involves understanding the system context, identifying the main system components and gaining an understanding of how they interact. It proceeds by performing tasks targeted at developing the conceptual design, eliciting the functional requirements and analysing the safety requirements attributed to the system to be developed.

3.2.1 System Context and Conceptual Design Issue

It has been observed that (via the LLD) the CE phase tasks are not systematic in deriving a consistent understanding of the system environment. Similar types

safety working groups such as the GSN User Group, MoD safety forums and the SBAC (Society of British Aerospace Companies).

of entry in the LLD indicates that often too many details are left vague or not investigated with enough rigour. The result of this issue is that the domain and interface descriptions are ambiguous, which results in problems later in the life cycle as different engineers interpret the environmental context features in different ways, which increases the risk of rework. The need to model aspects of behaviour in the Dem/Val phase does highlight some of these ambiguities, but many are not uncovered until the more detailed Use Case modelling that occurs during EMD. The later this issue is uncovered, then the more costly it is to remedy which increases project risk in terms of expense and schedule time. This is identified as our first issue:

> **[EnvirI1]:** Incomplete understanding of the system context increases the risk of rework.

3.2.2 Initial Functional Requirements Issue

The elicitation tasks involve assimilating requirements from the various stakeholders including the customer, those in the Company responsible for business objectives and those responsible for ensuring that necessary legislation is satisfied. The information sources includes customer documentation, discussions with the customer and interviews with the other stakeholders. At this stage a certain amount of overlap and contradiction is not unexpected, the idea being that these issues are resolved by the analysis work undertaken in the following Dem/Val and EMD phases. The main issue identified with these tasks (from inspection of the LLD) is that of missing a requirement or not completely capturing all the desired aspects of a requirement in line with the findings in [79]. The later a problem is uncovered, then the greater will be the cost of remedying it. This leads to our second issue:

> **[RequI2]:** Incomplete requirements increases risk of rework.

3.2.3 Attributed Safety Requirements

The initial IPDP safety work involves assimilating the contracted customer safety requirements and then performing safety analysis to see if additional hazards apply and to identify failures and behaviour modes that might result in safety issues. The task consists of assessing hazard lists derived using domain knowledge, analysing the issues and problems encountered on similar systems and performing brainstorming sessions to formulate an initial hazard list for the system to be developed.

It has been observed via the LLD that because the detailed models necessary to support a full preliminary safety analysis are not available until relatively late in the early life cycle phases with the existing IPDP, the safety engineering lags the rest of the development and can cause costly re-work when changes are required due to any discovered safety issues. A significant improvement to the safety aspects of IPDP could be effected by moving the safety analysis earlier in the life cycle, without compromising its effectiveness. This results in our third issue:

[LateI3]: Late effective safety analysis increases risk of rework.

Another issue concerns traceability, which is an important aspect of many critical development processes and has a number of forms. Safety standards such as DS 00-56 and guidelines such as DO-178B demand traceability from the top level requirements all the way through to the code that implements them, and require traceability to the evidence (test, code walkthrough etc.) that validates them. A lack of traceability makes it difficult to check that a requirement has been implemented adequately. This identifies another important issue, our fourth, that can be captured as:

[TraceI4]: Traceability incompleteness increases the risk of rework.

3.2.4 The SRR Gateway Review

A System Requirements Review (SRR) is used to validate that all the necessary tasks have been adequately completed and this covers: (a) determining that the system

requirements have been completely and properly identified, (b) reviewing that the mission requirements definition process has been correctly followed and (c) ensuring that the system certification and safety needs are appropriately considered from the outset. The latter includes confirming that a safety plan and safety case are being developed commensurate with the level of the safety requirements attributed to the system. The SSR acts as a gateway between the CE phase and the Dem/Val phase. The programme is only allowed to proceed to the next Dem/Val phase when all the items listed above are deemed to have been completed to a satisfactory standard.

One major issue that can occur during the SSR (as observed in the LLD) is the fact that certain tasks are not completed at all because they have been "tailored out" of the development process. During the initiation of a project, programmes are allowed to tailor the IPDP to fit in with their particular circumstances. This can result in tasks being removed that are deemed necessary by the SRR reviewers, resulting in re-work before the programme is allowed to pass SRR. This can be considered as part of a broader problem defined as our fifth issue:

[IncomI5]: Task incompleteness increases the risk of rework.

3.3 Demonstration/Validation (Dem/Val) Phase

The main purpose of the Dem/Val phase is the development of the system specifications and it consists of three tasks: (a) develop the system theory of operation, (b) develop the sub-system theory of operation, and (c) derive the system specifications. These tasks will now be described and issues identified earlier but also affecting this Dem/Val phase will be highlighted and any new issues introduced where appropriate. There are safety analysis activities associated with each of these tasks.

During Dem/Val the initial requirements are reviewed and where necessary transformed into more detailed requirements. These transformations tend to be informal

and text-based, but are validated through formal peer reviews which check issues concerning soundness, completeness and consistency. The goal is to produce a set of requirements that could be implemented.

3.3.1 System Theory of Operation

This task involves system requirements analysis, identification and analysis of the initial system architecture and the production of various models to support the analysis. The models tend to be informal, based on block diagrams supported by descriptive text that are focussed on satisfying a particular task - thus they only have a limited reasoning capability. Neither are the models particularly well integrated, as each is produced to satisfy a particular need as required. For example, many of the Company's systems have serial links to allow communications between units. These links are modelled by specialist "firmware engineers" (i.e. engineers with extensive experience in implementing communications networks and protocols at the hardware/software interface) to ensure that they have the necessary capacity to support the design and any growth requirements. The analysis and modelling of these links are checked by the project's Software Architect and the Lead System Engineer to ensure they are accurate and complete. However, they are not formally checked by the rest of the team, some of whom have to interact with and make use of these communication links. As noted, the modelling is informal and only the limited reasoning that derives from using the descriptive text is available. In the past this has allowed ambiguity and lack of detail in the descriptions to result in misinterpretations of the design concept, and thus relates to **[EnvirI1]** defined above. This identifies a need for stronger modelling capabilities, ideally supporting simulation so that the engineers that have to use the links can gain a better understanding of how they operate, their capabilities and their limitations.

The analysis and modelling work culminates in the production of a system theory

of operation document that is also descriptive text. The system theory of operation document is validated with the customer before embarking on the next set of tasks which involve more detailed architectural analysis and the development of the sub-system theory of operation.

A problem observed from the LLD for this and the next task was that of inconsistent completion of the sub-tasks associated with the task. The amount of work undertaken to complete a task was found to be variable, and in some cases the tasks were not completed, with items being poorly addressed or even omitted. Investigations indicated that often the task descriptions were high level and allowed a wide range of interpretation or engineers thought that other elements of the task had been covered elsewhere. This is another example of **[IncomI5]** Task incompleteness, introduced above.

3.3.2 Sub-System Theory of Operation

The sub-system theory of operation task involves further analysis and modelling and the development of the solution system architecture. The modelling in this sub-phase has similar issues to those recorded above. The resulting sub-system theory of operation document describes the main functional blocks, together with their interfaces and interactions at a reasonably detailed level such that the following requirements issues are identified: requirements functionality omitted or not completely covered; ambiguous or incomplete requirements; requirements inconsistencies and clashes. It has been observed that there is no support for iteration back to earlier tasks in the IPDP and the task of deciding where to iterate back to is exacerbated by the parallel and overlapping nature of some of the tasks. Being able to trace back through the design is another instance of issue **[TraceI4]** introduced in section 3.2.3.

3.3.3 System Specifications

Work then develops the specifications and sub-system design descriptions resulting in the Equipment Specifications (ES), the final Interface Requirements Specification (IRS) and the initial allocation of requirements (ASR) to software and/or hardware, including allocation of the safety requirements. These are all textual documents that have to undergo safety analysis as part of the IPDP work, but the LLD has recorded issues with the effectiveness of some of these analyses – which relates to issue **[LateI3]**.

The last part of this System Specifications task and before entry into the SDR, each specification requirement has a validation method defined for it, but there is no way of confirming that overall "the right system is being produced". That is, without the ability to simulate sets of requirements on a broader scale it is difficult to be confident that the system has the desired behaviour. In fact these issues could not be checked formally until the implementation phase. This increases the risk that the right system is not being produced and is part of issue **[RequI2]**.

3.3.4 Safety Analysis Discussion

As noted above, the Dem/VAL phase includes a number of safety analysis tasks. The abstract and textual descriptive nature of the source information available tends to motivate the use of FFA, which is the most appropriate form of analysis to use with the current IPDP because design instability and lack of design information in a suitable form means that the cost of running a HAZOPS is not justified at this point. Unfortunately, as noted by McDermid [90], FFA relies heavily on human skill, is judgemental and only identifies about 80% of the safety issues - information recorded in the LLD supports this. The initial Dem/Val FFA work for the *DC* system reported in Appendix A.3 is an example of this problem (related to issue **[LateI3]**). The need for a sufficiently strong CRC was not identified by the FFA,

even though this information was part of the *DS* environment information – as shown by the later, specific, serial link analysis which did identify the information and thus highlighted the issue. Thus an adequate safety analysis requires that the system environment is sufficiently understood (part of **[EnvirI1]**).

As discussed in section 2.2.2 an effective safety analysis requires an appropriate model and inspection of the overall IPDP process indicates that a suitable model to support a HAZOPS could be formed from the architectural information that is available, and from structural (e.g. interface and timing) information that could be derived with some re-ordering of the tasks. For example one of the IPDP safety tasks in this phase requires that a safety analysis is applied to the combination of the sub-system theory of operation and the system block diagram (section 3.3.2). Currently FFA is used, but inspection of the information available prior to SDR indicates that the sub-system theory of operation and the block diagram could be used as the basis for a HAZOPS model during Dem/Val as long as: the architectural analysis was used as the basis of the analysis model, extra information was collated concerning critical timings and sufficient knowledge concerning the interface to the environment (including potential adverse impact on this environment) could be collected. Currently HAZOPS is not used until the later EMD phase, and the ability to move it earlier in the life cycle is a mitigation for issue **[LateI3]**. This potential improvement to the IPDP safety analysis process is considered further in Chapter 5.

3.3.5 SDR Gateway Review

When Dem/Val is complete a System Design Review (SDR) is held to review the maturity of and assess the risks involved with the allocated requirements and specifications. The SDR also confirms that the safety and certification issues are adequately addressed, i.e. that all requirements have a reasonable validation method assigned and that all appropriate safety analysis has been undertaken. It has been

observed that the existing IPDP approach forms a collection of useful, but disparate, models. Each model performs an important task, but these models are neither well integrated nor easy to validate.

The SDR inspects the artefacts produced during the Dem/Val phase, considers their quality and validates that they are appropriate and fit for purpose before allowing the programme to proceed to the EMD phase. If any of these artefacts are incomplete then rework must be done. However, it has been observed from the LLD that projects that have incomplete validation criteria applied to the requirements encounter significant problems later in the development life cycle. This indicated the need for improved measures to ensure that the validation criteria assigned to each requirement is complete. Therefore another important issue, our sixth, can be captured as:

[ValidI6]: Validation incompleteness increases the risk of rework.

3.4 EMD – Preliminary Design Phase

The Preliminary Design sub-phase is the first set of activities in the EMD phase, and is the only phase of EMD discussed in this document. It uses the documents emanating from the SDR and starts with a detailed analysis of the customer requirements and their allocations to the software, the hardware or both. The architecture is developed and it also produces high level design descriptions to feed into the following more detailed design phases (not considered here). As part of this process the requirements are analysed in more detail and developed to support the evolving design. A variety of safety analysis tasks are conducted during the Preliminary Design phase.

3.4.1 Requirements Allocation and Architecture Design

Requirements are allocated based on hardware/software trade-off activities and all designs, analyses and evaluations are documented and all applicable specifications are updated before proceeding with the next tasks. The design task then splits into its software, hardware and system components, which are handled by the appropriate development teams.

The architectural design refines and develops the system architecture produced during Dem/Val, to add more detailed functional and interface descriptions. This process also identifies derived requirements reflecting the design decisions that are made to support the architectural development.

It has been observed from the LLD that the detailing of the requirements allocations and transformations do not always produce requirements that are feasible, nor are they necessarily sound, consistent or complete. Peer reviews are conducted and these resolve many issues, but some problems can remain – this relates directly to **[RequI2]**.

3.4.2 Preliminary Design Tasks

The preliminary design tasks are Preliminary System Design, Preliminary Software Design, and Preliminary Hardware Design. They all involve analysis of the requirements allocated to their discipline in conjunction with a more detailed architectural design (system, software and hardware) . For example, the purpose of the software analysis and detailed architectural design activities are to define how the software will realise the requirements on it, and to define a stable software architecture based on the Object Oriented principles [13, 56] used in the development. All designs, analyses and evaluations are documented and all applicable specifications are updated before proceeding with the design. A similar process is followed by the other disciplines.

Detailed design is not allowed to be started until after successful completion of the PDR checkpoint gateway review. Failure to observe this rule on some previous projects (as recorded in the LLD) has resulted in expensive rework – the causes being instances of our six issues identified above.

The hardware development follows a single pass strategy, but it is typical to run the software development over a number of iterations. The first iteration de-risks any software architecture concerns by providing a prototype whose behaviour can be evaluated, and if successful this iteration is used as the basis for the subsequent iterations. Otherwise a viable alternative is produced, informed by the problems identified with the original architecture (system and/or software) . This has cost and schedule implications for the development as IPDP does not easily support an iterative loop structure (it has been specifically updated to "hard code in" the software iterations), any identification of problems causes a dislocation in the schedule which is scrutinised by senior management - hence there is pressure to avoid iteration and adopt a "patch and proceed" mentality. The ability to analyse the architecture earlier in the development cycle and the ability to manage iteration have been identified as two significant enhancements to the capability of the IPDP. This demonstrates the need for direct support for iteration, which is a form of traceability issue and thus is part of **[TraceI4]**.

At this point in IPDP a higher integrity process is used for system developments which have functional safety requirements that involves:

- the use of special safety architectures (system, software and hardware) which include protective and defensive features;
- the specifications being written in the Z formal method [116] to define the safety behaviour;
- the use of proof to show that the implemented code satisfies the Z specification.

3.4.3 EMD Preliminary Design Safety Analysis

The EMD Preliminary Design phase includes eight safety analysis tasks including reviews to cover health and safety and the legislative aspects. For developments with functional safety requirements a modified form of HAZOPS is applied to the developed architecture, using the allocated requirements and detailed descriptions of the context derived from the earlier tasks. The analysis begins around PDR and yields good results (no issues recorded in the LLD), albeit rather later in the process than desired – hence it is related to **[LateI3]**.

3.4.4 PDR Gateway Review

The Preliminary Design phase is exited after successful completion of a Preliminary Design Review (PDR). The role of the PDR is to assess that the maturity of the design is appropriate and to approve initiation of the detailed design. It involves formally authenticating all the Development Specifications and releasing them as approved. Further goals are to evaluate the technical adequacy and risk management of the selected design approach, and to confirm that the certification and safety aspects are apposite.

The successful completion of the PDR means that an appropriate design specification, S, has been produced, and inspection of the LLD shows that the majority of projects achieve this goal. Therefore a necessary property of any approach that is to be used in conjunction with IPDP is that it should be at least as good as the current IPDP at deriving a design specification. This is captured as the necessary property (NecessaryP):

> **[NecessaryP]:** The approach used must be capable of deriving the design specification.

Any approach which does not satisfy **[NecessaryP]** cannot be considered for use with IPDP.

A factor influencing why the safety team lags behind the other disciplines (refer to section 3.2.3) is the need to produce a specific safety model to support the safety analysis work and the task of keeping it up to date. If the safety analysis used the same model as the other disciplines then this would be much more efficient and less error prone (no need to validate the safety model). Therefore an advantageous property of an approach can be stated as follows:

> **[SameA1]:** The safety analysis uses the same model as the rest of the development.

For developments with functional safety requirements IPDP follows an augmented process that includes the use of formal specification and proof techniques, therefore it is highly desirable that any approach used should be capable of supporting formal development processes. This is captured as the second advantageous property:

> **[FormalA2]:** The approach used should support formal development processes.

3.4.5 Summary of Issues and Properties

In general the Company has found IPDP to be an excellent mechanism for organising and running projects. It promotes completeness and consistency across the projects and has reduced variation significantly. The experience gained from running many projects through IPDP has resulted in improvements to the process to support how the Company organises its work (e.g., inclusion of an iterative software process) and this improvement is an ongoing process. However, as discussed in the previous sections, some significant issues have been observed which the following text summarises.

It is worth noting that these issues and properties apply to safety development processes other than IPDP. That is, the issues and properties identified in the preceding sections are important for any safety process as well as for the IPDP used for safety development and the identified issues are:

1. **[EnvirI1]:** Incomplete understanding of the system context increases risk of rework.

2. **[RequI2]:** Incomplete requirements increases risk of rework.

3. **[LateI3]:** Late effective safety analysis increases risk of rework.

4. **[TraceI4]:** Traceability incompleteness increases risk of rework.

5. **[IncomI5]:** Task incompleteness increases risk of rework.

6. **[ValidI6]:** Validation incompleteness increases risk of rework.

The identified properties are:

1. **[NecessaryP]:** The approach used must be capable of deriving design specification S.

2. **[SameA1]:** The safety analysis uses the same model as the rest of the development.

3. **[FormalA2]:** The approach used should support formal development processes.

Any approach introduced to improve the capabilities of an existing process, IPDP in this case, must integrate into this existing process. It must address at least some of the issues identified with the existing process and it must be straightforward to implement, as undue complexity could result in resistance and it not being used effectively. This need for good integration can be added to the list of advantageous properties as follows:

[IntegA3]: The approach used should integrate with IPDP.

In the remainder of this thesis we show how POSE can be used to address these six identified issues whilst satisfying the properties identified so far (i.e., the ten properties from section 2.5 and the four identified in this chapter). Specifically, Chapter 4 introduces POSE in its basic, "vanilla" form and evaluates its capabilities in dealing with the identified issues and properties. The chapters that follow develop POSE to address any outstanding issues and properties.

Chapter 4

Supporting the Process Using POSE

The aim of this chapter is to investigate whether POSE could be successfully used within the IPDP (addressing property **[IntegA3]** of Chapter 3), and to which extent in its "vanilla" formulation, as defined by [39, 40, 41], it would be able to address the properties and issues summarised in section 2.5 and section 3.4.5.

Throughout the chapter we make use of a case study, the Decoy Controller (*DC*), to introduce POSE and its features, illustrate its application, and evaluate its potential for real-time embedded avionics safety systems. The *DC* case study refers to a real development problem at the Company, which was originally addressed using the IPDP. It was also used by the author in [87] to introduce the capabilities of POSE in the context of embedded avionics applications. Here it is investigated in more detail in the context of working within the IPDP.

4.1 Introduction to Problem Solving with POSE

Our first task is to introduce the Decoy Controller *DC* case study that will form the basis for the rest of this chapter and will also be used in the initial part of Chapter 5.

4.1.1 Decoy Controller Case Study Description

The Decoy Controller *DC* system's main components are now introduced (a more complete representation of the case study and its use of IPDP is given in Appendix A). The DC is part of the defensive aids suite (DAS) on an aircraft and its block diagram is shown in Figure 4.1. The DAS is controlled by a Defence System (DS) LRU (Line Replaceable Unit) which sends control and information messages to the DC over a serial link. The DC controls the selection and release of flares from the Dispenser Unit (DU) based on control inputs it receives from the DS, the pilot and the aircraft status. The latter indicates if the aircraft is on the ground or in the air.

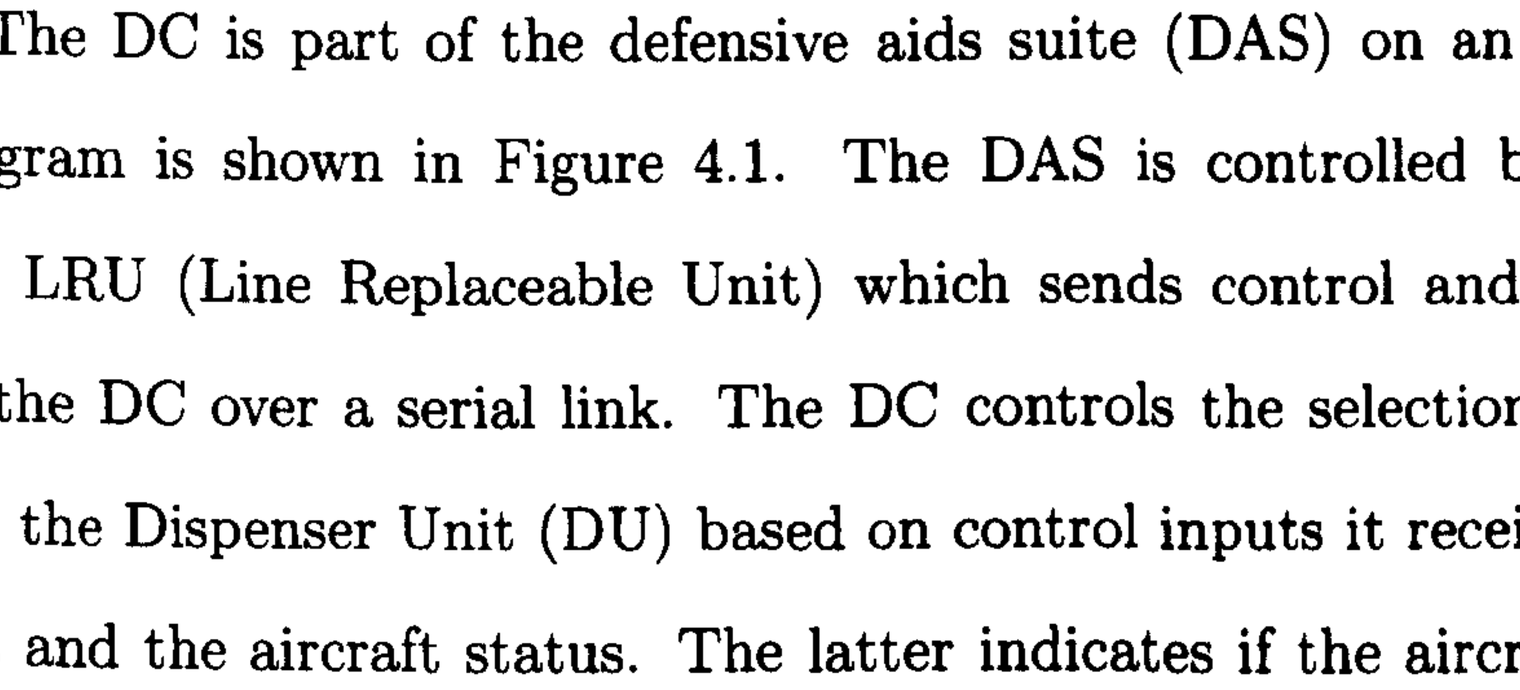

Figure 4.1: Decoy Controller (DC) Block Diagram

The IPDP CE phase safety work was conducted on the conceptual model formed from the system block diagram (see Figure 4.1), the system description (derived from the work completed to date) and the system requirement *RDC* – as described in Appendix A.2. The requirement *RDC* was the original requirement as used on the project, it does not reflect current good practice based on using the DOORS tool to capture the requirements and their traceability [118]. In particular it does not define only one requirement at a time and includes multiple instances of using conjunctions that make multiple requirements in a single statement. The reason why we use the "original" requirement was to show that POSE could be used successfully with requirements that might be encountered on actual projects and with requirements

that were not *doctored* to make then easier to use. A re-worked *DC* case study based on using [118] to re-structure the requirements into a good practice form is shown in Appendix H.

The requirements were further refined and detailed by tasks completed during the Dem/Val phase (refer to section 3.3) to form *Rdc* as detailed in Appendix A.3. The analysis work for the Dem/Val phase, also recorded in Appendix A.3, identified a number of safety issues and process problems when using the task descriptions in the existing IPDP process. For example, in the DAS the *DC* has to communicate with the Defence System (*DS*) over a serial link. The FFA safety analysis identified that a high integrity algorithm should be used to obtain the required integrity for the message passing function. Unfortunately, this was misinterpreted as the standard parity and message block system used for most of the Company's serial link implementations. The specific requirements that demanded the use of a Cyclic Redundancy Check (CRC) algorithm were not assimilated and the problem was not discovered until a specific serial link safety analysis identified the integrity shortfall. As in this case, any such issues are identified during the system analysis and integration phases, but these are well into the development life cycle where changes are costly and have a negative impact on the schedule. Therefore, as noted above, there is an identified improvement to the IPDP in trying to capture such issues earlier in the life cycle – this is issue **[LateI3]** defined in Chapter 3.

The EMD Preliminary Design phase work of Appendix A.4 shows how the final system architecture for the *DC* system was derived. It also notes that this system architecture was chosen because it was based on an existing prototype that was known to be capable of satisfying the functional requirements, and it is typical of industrial safety design strategies that attempt to minimise the number and extent of the safety related functions by localising them to simple, distinct blocks.

In what follows we revisit the *DC* development problem detailed in Appendix A, but this time applying POSE as appropriate to provide specific support to the

relevant IPDP tasks. The case study is cut down only in the sense that some detail has been removed for brevity: it retains all essential complexity.

At this point it is useful to recall briefly all the phases involved in the process of eliciting the requirements. As part of the successful contracting process the mechanical outline, approximate weight and power envelope of the system was established. Further, the initial high level functional requirements, **RA - RD**, were agreed as defined in section A.2. Subsequent communications with the customer were used to clarify the requirements and properties of the system environment and resulted in agreement that **RS** defined the safety requirements attributed to the system from the aircraft and system level hazard analyses. All these requirements formed the basis under which the remainder of the system was developed.

4.1.2 Software Problems in POSE

Hall, Rapanotti and Jackson [39, 40, 41] introduce a software problem as having three elements: a real-world context, W, a requirement, R, and a solution, S, which are related by the entailment $W, S \vdash R$.

The problem context is a collection of *domains* ($W = D_1, ..., D_n$) described in terms of their known, or *indicative*, properties, which interact through their sharing of *phenomena* (i.e, events, commands, states, *etc.*. [53]). More precisely, a *domain* is a set of related phenomena that are usefully treated as a behavioural unit for some purpose. A domain can be a unique, physical entity, or it can be an aggregate of physical entities that are combined to achieve a specific effect – the selection will depend on the context of the problem being solved.

In POSE all problem solving starts from the *null problem*, the problem of which we know nothing other than it exists [41]:

$$P_{null}: \quad W : null, S : null \vdash R : null$$

null is used as the description for W, R and S to indicate that nothing is known about them: *null* has less information than any description that can be written in any language chosen for descriptions.

4.1.2.1 Domain Context in POSE

The environment context is described in terms of domains. A domain $D(p)_o^c = N : E$ has *name* (N) and *description* (E), the description indicating the possible values and/or states that the domain's phenomena (in $p \cup c \cup o$) can occupy, how those values and states change over time, how phenomena occur, and when. Of the phenomena: c are those *controlled* by D, i.e., visible to, and shareable by, other domains but whose occurrence is controlled by D; o are those *observed* by D, i.e., made visible by other domains, whose occurrence is observed by D; p are those *unshared* by D, i.e., shareable by no other domain. The p are often termed internal phenomena and can be omitted if they have no impact on the development.

As an example of the first step in using POSE consider the *DC* case study. This first step involves collecting information about the existing environment which the proposed system is to operate in. Initially this information will just be an identification of the domains involved, but this will be enriched by further investigation and analysis which will ensure a detailed understanding of the environment is amassed that allows the appropriate domains to be identified, together with a knowledge of the properties that they have and an appreciation of how they interface to each other. It maps well with the system environment and initial requirements tasks in the CE phase (see section 3.2). The approach is in line with the "understand the environment before the new system is applied ideas" introduced in section 2.4 which is an important aspect of the problem oriented methods and thus supports property **Context** from Table 2.3. The domains in the *DC* case study are drawn from the entities represented in Figure 4.1 and comprise of the pilot *Pilot*, Defence System *DS*, Dispenser Unit *DU*, Safety Pin status *SP*, the Aircraft Status *AS* and

the Decoy Controller *DC* domains. Knowledge about these domains is increased by investigating their respective interface phenomena, the results of this work are shown in Table 4.1 on page 82.

4.1.2.2 Requirements in POSE

A problem's requirement states what is required of the problem solution. Like a domain, a requirement is a named description with phenomena, $R^{cons}_{refs} = N : E$. A requirement description should always be interpreted in the optative mood, i.e., as expressing a wish. As to the requirement's phenomena: *cons* are those *constrained* by R, i.e., whose occurrence is constrained by the requirement, and whose occurrence the solution affects in providing a solution; *refs* are those *referenced* by R, i.e., whose occurrence is referred to but not constrained by the requirement. Knowledge about the requirements is assimilated and organised in terms of the identified domain structure and interfaces. This means the requirements are reviewed and articulated into an appropriate form such that they are written in terms of the problem space as a customer would understand them, i.e. in terms of the environment domains. Care is taken to ensure these requirements avoid any implementation bias, and this supports property **Avoid Bias** from Table 2.3.

As an example, in the *DC* case study the various information sources – the customer requirements, customer meeting minutes and email communications – were reviewed in-depth to support this work and this allowed the initial problem to be developed (supporting properties **Model Reality** and **Validation** from Table 2.3) as follows. The requirements were refined and detailed from *RDC* to $Rdc = Ra \wedge Rb \wedge Rc \wedge Rd \wedge RS$ as defined in section A.3 and the requirements represented by *Rdc* are repeated below:

Ra The *DS* shall command which flare is selected using a field in its *con* message issued to the *DC*. The *DC* shall obtain the selected flare information from this

field in the *con* message, and use it in its *sel* message to the *DU* to control the flare selection in the *DU*.

Rb The *DS* shall command the *DC* to issue a *fire* command in its *con* message. This shall be the only way in which a flare can be released.

Rc The *DC* shall cause a flare to be released by issuing a *fire* command to the *DU*, which will fire the selected flare.

Rd The *DC* shall only issue a fire command if its interlocks are satisfied, i.e. aircraft is in air ($AS = on$), *SP* safety pin has been removed ($SP = off$) and *Pilot* has issued an allow a release command ($Pilot = on$).

RS The *DC* shall mitigate **H1** and **H2** (Target: safety critical 10^{-7} fpfh); where **H1** and **H2** are defined in Appendix A.2.

4.1.2.3 Solution in POSE

From [39], a software solution is a domain, $S(p)^{c}_{o} = N : E$, that is intended to solve a problem, i.e., when introduced into the problem context will satisfy the problem's requirement. The possible descriptions of a solution range over many forms, from high-level specification through to program code. As a domain, a solution has controlled, observed and unshared phenomena; the union of the controlled and observed sets is termed the *specification phenomena* for the problem. In the initial description of the case study problem the solution domain is *DC*.

A problem's elements come together in POSE in a *problem sequent*[39]:

$$P:\ D_1(p_1)^{c_1}_{o_1}, \ldots, D_n(p_n)^{c_n}_{o_n}, S(p)^{c}_{o} \vdash R^{cons}_{ref}$$

where the name P is optional. By convention (e.g. see [41]), the problem's solution domain, S, is always positioned immediately to the left of the $\vdash$. Note that the

descriptions of a problem's elements may be in any language, different elements being described in different languages, should that be appropriate [39].

As an example of a Software Problem return again to the *DC* case study. Problem solving starts with the null problem ([41], page 230), P_{null}, and the target of the first phase of applying POSE is to form the initial problem, $P_{Initial}$, based on Figure 4.1. Eventually this will have the form:

$$P_{Initial}: \quad DS(con)^{con},\ DU(fire, sel)_{fire,sel},\ SP(out)^{out},$$
$$AS(air)^{air},\quad P(ok)^{ok},\quad DC^{fire,sel}_{ok,air,con,out} \quad \vdash \quad Rdc^{fire,sel}_{con,out,air,ok}$$

However, before this can be derived we have to introduce a number of POSE transformations.

4.1.3 POSE Problem Transformations

Problem transformations capture discrete steps in the problem solving process [35]. Many classes of transformations are recognised in POSE, reflecting a variety of software engineering practices reported in the literature or observed elsewhere [39]. Problem transformations relate a problem and a justification to (a set of) problems. Problem transformations conform to the following general pattern. Suppose we have problems $W, S \vdash R$, $W_i, S_i \vdash R_i$, $i = 1, ..., n$, $(n \geq 0)$ and justification J, then we will write:

$$\frac{W_1, S_1 \vdash R_1 \qquad ... \qquad W_n, S_n \vdash R_n}{W, S \vdash R}\ \begin{matrix}[\textsc{Name}] \\ \langle\!\langle J \rangle\!\rangle\end{matrix}$$

to mean that, derived from an application of the NAME problem transformation schema (discussed below):

S is a solution of $W, S \vdash R$ with *adequacy argument* $(CA_1 \wedge ... \wedge CA_n) \wedge J$ whenever $S_1, ..., S_n$ are solutions of $W_1, S_1 \vdash R_1, ..., W_n, S_n \vdash R_n$, with adequacy arguments $CA_1, ..., CA_n$, respectively.

This transformation allows $W, S \vdash R$ to be replaced with $W_1, S_1 \vdash R_1, ..., W_n, S_n \vdash R_n$ with the justification given by J. The idea being that each $W_i, S_i \vdash R_i$ from $W_1, S_1 \vdash R_1, ..., W_n, S_n \vdash R_n$ is easier to solve than the original $W, S \vdash R$. This notion of moving from difficult problems to easier to solve sub-problems is an important aspect of the POSE approach, and supports the property **Simplification** from Table 2.3. POSE problem transformations with appropriate examples are explained in detail in [39].

Therefore, software engineering design under POSE proceeds in a step-wise manner: the initial "null" problem forms the root of a *development tree* with transformations applied to extend the tree upwards towards its leaves [35]. Branches are completed by problem transformations that leave the empty set of premise problems. Appropriate transformations applied to the null problem transform it to represent an initial form of the software problem to be solved - the initial POSE requirement model. This involves instantiating knowledge about the environment domain context (to address **[EnvirI1]**), the requirement (to address **[RequI2]**) and to introduce the solution domain. The environment domain context information is introduced using the CONTEXT INTERPRETATION transformation ([41], page 228) and the acquired requirements knowledge is introduced using the REQUIREMENTS INTERPRETATION transformation ([41], page 233). The software solution domain is introduced using the SOLUTION INTERPRETATION transformation ([41], page 234), which makes use of the *Architectural Structure AStruct*[...] ([41], page 236) to introduce known solution components and the solution components to be designed – the *AStruct*[...] is considered in more detail in section 4.1.3.3 below.

Table 4.1: Phenomena of the *DC* Problem

Phenomenon	Designation
fire	Command to release the selected flare
sel	Indicates which flare type should be selected
out	Pin status; $out = yes$ indicates pin has been removed
con	Contains command to fire and selected flare type
air	Aircraft status; $air = yes$ indicates aircraft is in the air
ok	Pilot intention; $ok = yes$ indicates allow release

4.1.3.1 Introducing Context in POSE

As an example of the use of introducing the context into the software problem return to the *DC* case study and the information in section 4.1.2.1. The initial environment context work includes the description of *Sys* on page 228, which provides an overview that introduces the individual domains that comprise the complete system of interest. These domains are *Pilot*, *AS*, *SP*, *DS* and *DU*. This basic information allows the software problem description to be enhanced from P_{null} about which nothing is known to P_{doms} where the domains involved are identified using the CONTEXT INTERPRETATION transformation as follows:

$$\frac{P_{doms}: \ W : [P : null, AS : null, SP : null, DS : null, DU : null], S : null \vdash R : null}{P_{null}: \ \ W : null, S : null \vdash R : null} \ \begin{matrix}[\text{CI}] \\ \langle\!\langle \text{JC} \rangle\!\rangle\end{matrix}$$

where CI is CONTEXT INTERPRETATION and the justification JC is that context analysis has identified the listed domains as being relevant to the problem

Each of these context domains is then investigated in detail to understand its behaviour, note any concerns about its operation and identify its phenomena (supporting the IPDP theory of operation tasks). The descriptions *Pilot* through *DU* are those listed on page 228, and those are the relevant parts of the domain knowledge gathered from the information sources.

The phenomena associated with these domains are defined in Table 4.1. The

fire and *sel* phenomena are inputs to the DU and the representation of the DU in POSE is given by $DU_{fire,sel}$. The SP provides the status of the pin *out* as an output and in POSE this is represented as SP^{out}. This means that the SP term in the requirement **Rd** of $SP = off$ maps to the phenomena $out = yes$, and $SP = on$ maps to $out = no$. Similarly the AS provides the in air status as an output using *air* ($air = yes$ maps to $AS = on$ in **Rd**), and the *Pilot* sends the switch selection using *ok* ($ok = yes$ maps to $Pilot = on$ in **Rd**). In POSE these are respectively represented as AS^{air} and $Pilot^{ok}$. The *con* command message is an output from the $DS - con$ is also used in *Rdc* so no mapping is required. In POSE this is represented as DS^{con}. The mappings defined above show that the model is grounded in reality (supporting property **Model Reality** from Table 2.3).

This increased knowledge is then introduced into the software problem using a series of CONTEXT INTERPRETATION transformations – as described in [41] pages 232 to 233 – to form the software problem P_{Dom}. In this software problem, P_{Dom}, and those that follow we will often omit the denotations "W :", "S :" and "R :" whenever they can be inferred by context – in P_{Dom} the "W :" is omitted.

$$P_{Dom}: \ DU_{fire,sel},\ SP^{out},\ DS^{con},\ AS^{air},\ P^{ok},\ S : null \ \vdash \ R : null$$

The collective justification for these CONTEXT INTERPRETATION transformations (J_a) is that the analysis of the information sources identified that the environment consisted of a DS defence system that can send command messages to the DC which can then control the release of flares from the DU taking due account of the inputs received from the *Pilot*, AS and SP domains. The information sources were then peer reviewed to confirm and validate that these were the only significant environment domains providing evidence for [**EnvirI1**] and property **Context** from Table 2.3.

4.1.3.2 Introducing Requirements in POSE

Next the requirements are introduced into the problem model using the REQUIREMENTS INTERPRETATION transformation ([41], page 233). As detailed in Appendix A.2, the initial customer requirements were represented by requirement *RDC* and these requirements were refined and detailed as part of the IPDP CE and Dem/Val phase work to produce the final requirement *Rdc*, which is listed in full above in section 4.1.2.2. In POSE this refining and detailing is achieved using the REQUIREMENTS INTERPRETATION transformation. An example will illustrate the process followed. One of the original requirements from *RDC* concerned the messages sent by the *DS* and had the form:

RA The *DS* shall command which flare is selected for release by using a field in the *con* message it transmits.

This is refined and detailed by the analysis of the initial system architecture shown in Figure 4.1, conducted as part of the tasks in the Dem/Val phase, to form the more detailed requirement:

Ra The *DS* shall command which flare is selected using a field in its *con* message issued to the *DC*. The *DC* shall obtain the selected flare information from this field in the *con* message, and use it in its *sel* message to the *DU* to control the flare selection in the *DU*.

Similar detailing was applied to the other requirements in *RDC* to form *Rdc*, and these can be introduced into the problem using a series of REQUIREMENTS INTERPRETATION transformations. That is, the step from P_{Dom} into $P_{Dom\&R}$ which introduces the requirement *Rdc* into the problem is actually a sequence of REQUIREMENTS INTERPRETATION transformations – a similar arrangement is detailed in [41] for the refinement and detailing of the Package Router requirements.

$$P_{Dom\&R}: \quad \begin{array}{l} DS(con)^{con},\ DU(fire, sel)_{fire,sel},\ SP(out)^{out}, \\ AS(air)^{air}, \quad P(ok)^{ok}, \quad S : null \end{array} \quad \vdash \quad Rdc^{fire,sel}_{con,out,air,ok}$$

where the analysis included identifying the requirements phenomena *cons* and *ref*, which for the *DC* case study are defined by the following:

$cons = \{fire,\ sel\}$

$ref = \{out,\ air,\ ok,\ con\}$.

The justification for this REQUIREMENT INTERPRETATION transformation (J_b) is that the $Rdc = Ra \wedge Rb \wedge Rc \wedge Rd \wedge RS$ were validated against the information sources and found to be adequate representations of the intended behaviour of the system and were confirmed as apposite by the customer, providing evidence to support **[RequI2]** and property **Validate** from Table 2.3.

4.1.3.3 Introducing Solutions in POSE

The Architectural Structure, *AStruct* [41], is used to initially introduce the solution domain, but it is also used to add structure to this solution domain, through an application of SOLUTION INTERPRETATION. That is, it can be used to introduce different solution architectures into the POSE requirements model. An *AStruct* combines, in a given topology, a number of known solution components[1] (the C_i below) with solution components yet to be found (the S_j below). Its general form is:

$$AStructName[C_1, ..., C_m](S_1, ..., S_n)$$

[1]There are also constraints on the phenomena sets, which we omit here for brevity, refer to [39] for the full definition.

with *AStructName* the *AStruct* name. Once the solution is interpreted by providing and justifying an *AStruct*, SOLUTION EXPANSION generates premise problems by moving the already known components C_i to the environment—expanding the problem context—whilst simultaneously refocussing the problem to be that of finding the solution components S_j that remain to be designed. The requirement and context of the original problem is propagated to all sub-problems.

A particular case, which is relevant to the case studies, is when there is only one component to be found, that is, the *AStruct* has the following form:

$$AStructName[C_1, ..., C_m](S)$$

In this case expansion only generates one premise problem as follows:

$$\frac{\mathcal{W}, C_1, ..., C_m, S : null \vdash R}{\mathcal{W}, S : AStructName[C_1, ..., C_m](S) \vdash R} \text{ [SOLUTION EXPANSION]}$$

An example of an *AStruct* is used to introduce the solution domain, DC, into the software problem as follows. At this point we know that DC is the domain to be designed and there are no known components. The description of the DC is listed in the IPDP:Dem/Val Phase subsection of section A.3, and analysis identifies its interrelationships and the phenomena it shares with the existing domains, such that it can be written as $DC^{fire,sel}_{ok,air,con,out}$. The specification to be found of the DC is introduced through SOLUTION INTERPRETATION using an *AStruct* ([41], page 236) of the form:

$$S : DCS[](DC)^{fire,sel}_{ok,air,con,out}$$

The justification for this solution interpretation step (J_c) is that it just introduces the solution domain, DC, into the software problem. Further, similar system architectures have been used successfully on other aircraft recording the fact that the overall architecture represented by P_{DomRS} has a good track record. The SOLUTION INTERPRETATION transformation ([41], page 234) is applied to transform the

software problem from $P_{Dom\&R}$ into P_{DomRS}.

$$P_{DomRS}: \quad \begin{array}{l} DS(con)^{con},\ DU(fire,sel)_{fire,sel},\ SP(out)^{out}, \\ AS(air)^{air},\quad P(ok)^{ok},\quad S : DCS[](DC)^{fire,sel}_{ok,air,con,out} \end{array} \quad \vdash \quad Rdc^{fire,sel}_{con,out,air,ok}$$

The next step is to expand $S : DCS[](DC)^{fire,sel}_{ok,air,con,out}$ using SOLUTION EXPANSION to transform the problem from P_{DomRS} into $P_{Initial}$ as follows:

$$P_{Initial}: \quad \begin{array}{l} DS(con)^{con},\ DU(fire,sel)_{fire,sel},\ SP(out)^{out}, \\ AS(air)^{air},\quad P(ok)^{ok},\quad DC^{fire,sel}_{ok,air,con,out} \end{array} \quad \vdash \quad Rdc^{fire,sel}_{con,out,air,ok}$$

Inspection of the process so far shows that the problem derivation is problem oriented and does not involve the introduction of any detailed structures from the solution space. This avoids solution bias issues, providing support for property **Avoid Bias** from Table 2.3.

4.1.4 Problem Solving Steps

4.1.4.1 Problem Transformation Schema and Templates

The general form of a POSE problem transformation schema was introduced at the beginning of section 4.1.3, and an example of CONTEXT INTERPRETATION presented in section 4.1.3.1. The REQUIREMENTS INTERPRETATION and SOLUTION INTERPRETATION problem transformation schema have a similar form to that of the CONTEXT INTERPRETATION schema. The justification associated with each

problem transformation schema can vary in strength from a vague notion through to an extremely strong and compelling linkage [39, 40]. In general the latter is preferred, but is not essential – it depends on the context, e.g formal proof might be required for safety critical developments, whilst weaker evidence will be sufficient for less critical ones.

Each transformation has associated with it a number of elements that are an important part of its structure and the transformation must be viewed in conjunction with these elements. For example, some of the transformations in [41] include *Concerns* and/or *Phenomena* descriptions as well as the justification – it would be incorrect to present just the transformation part without these elements. We will now present a template structure that allows the transformation and its elements to be conveniently collected into a single unit, with the additional advantage that it separates them from the narrative (following our work in [35]). Suppose we wish to transform the problem $P(= W, S \vdash R)$ under the NAME transformation schema, then we will write [35, 41]:

Application of NAME *to problem P*

JUSTIFICATION J: describing the named justification (J) for the application of transformation NAME to P. The name can be used elsewhere in the development to refer to this justification. The body of the justification can have many components—prose, formal descriptions, figures, for instance—and any or all elements of the following structure may be present:

INCLUDES: identifying any relationship between this justification and others in the development such as those that occurred from an earlier step, that was subsequently discovered to be inadequate and so backtracked from. The inadequacy can also be described here.

CONCERNS: A collection of standard concerns associated with each transformation

schema. Each concern is discharged by argument and evidence supporting a claim of the following form:

CLAIM: *Claim statement*

ARGUMENT & EVIDENCE: The reason to believe the claim, or the reason it does not hold.

PHENOMENA: Should the schema introduce phenomena, or need to detail their sharing, the details can be included here.

Note that this transformation template construction has used concepts from previous work including "Concerns" from Problem Frames [53] and the use of "Claims, Argument, Evidence" ideas in a similar way to Adelard's ASCE tool [12].

4.1.4.2 Problem Transformation Steps

The individual POSE transformations operate at a detailed level and a number have to be combined in sequence to achieve a useful step in the development process. For example, the formation of $P_{Initial}$ in section 4.1.3.3 involved a series of interpretation and expansion transformations. Inspection of the use of POSE in the literature [41, 87, 86, 85, 88] indicates that this formation of an initial software problem is a standard practice and it is a useful extension of the basic POSE approach to capture this sequence, and others like it, as a distinct entity. In this work the collection of a series of POSE problem transformations into a useful step in the development process will be called a *Problem Transformation Step* and in accordance with the original introduction of the template structure in [35] we will use a template structure to capture each problem transformation step.

4.2 POSE Support for IPDP

The aim of this section is to show how the POSE transformations can assist in supporting the IPDP safety process, although the concepts can equally be applied to other similar safety development processes, by ascertaining to what extent they can address the properties and issues identified in section 2.5 and section 3.4.5, using the ideas and notation introduced in section 4.1.3.

At this point in the case study the context and requirement have been established as part of the IPDP CE phase work, the requirement developed, and the *AStruct* has been used to establish the initial system architecture (early part of IPDP Dem/Val). Therefore the transformation sequence involved in forming the initial software problem, $P_{Initial}$, can be summarised by the following POSE transformation sequence, where $P_{Initial}$ is the top line of the sequence.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{DU(fire,sel)_{firc,sel}, SP(out)^{out}, DS(con)^{con}, AS(air)^{air}, P(ok)^{ok}, DC^{fire,sel}_{ok,air,con,out} \vdash Rdc^{cons}_{ref}}{DU(fire,sel)_{fire,sel}, SP(out)^{out}, DS(con)^{con}, AS(air)^{air}, P(ok)^{ok}, S : DCS[](DC)^{fire,sel}_{ok,air,con,out} \vdash Rdc^{cons}_{ref}}\,[SolutionExp.]}{DU(fire,sel)_{firc,sel}, SP(out)^{out}, DS(con)^{con}, AS(air)^{air}, P(ok)^{ok}, S : null \vdash Rdc^{cons}_{ref}}\,[SolutionInt.]}{DU_{fire,sel}, SP^{out}, DS^{con}, AS^{air}, P^{ok}, S : null \vdash R : null}\,[RequirementsInt.]}{W : [P : null, AS : null, SP : null, DS : null, DU : null], S : null \vdash R : null}\,[5xContextInt.]}{P_{null} : \quad W : null, S : null \vdash R : null}\,[ContextInt.]$$

Substituting in for *cons* and *ref* (see page 85) allows $P_{Initial}$ to be re-written as:

$$P_{Initial} : \begin{array}{l} DS(con)^{con}, DU(fire,sel)_{fire,sel}, SP(out)^{out}, \\ AS(air)^{air}, \quad P(ok)^{ok}, \quad DC^{fire,sel}_{ok,air,con,out} \quad \vdash \quad Rdc^{fire,sel}_{con,out,air,ok} \end{array}$$

This problem, $P_{Initial}$, can also be represented as an alternative view using the PF problem diagram as shown in Figure 4.2. The model represented by $P_{Initial}$ and

Figure 4.2 corresponds to the output of the System and Sub-System Theory of Operation tasks in the IPDP Dem/Val phase (refer to section 3.3 and Appendix A.3).

The series of transformations followed in developing $P_{Initial}$ from P_{null} consisted of gaining knowledge and understanding of the environment context, the requirements, the system architecture and how they all interact. This sequence covers a number of the IPDP tasks in the CE and Dem/Val phases as discussed above, and is typical of those steps required in the initial phases of all developments as discussed in section 4.1.4.2. Therefore, these POSE transformations can be combined into the **PS1 Initial Problem** problem transformation step represented by the structure shown below.

PS1: ***Initial Problem*** *application to transform problem* P_{null} *into* $P_{Initial}$

JUSTIFICATION J_1: The identified requirement, domains and their relevant properties are summarised as defined in sections A.1 – A.3 and in the development of $P_{Initial}$ above and $J_1 = J_a \wedge J_b \wedge J_c$.

PHENOMENA: Phenomena and their control and sharing (see $P_{Initial}$) are known from the existing system components as shown in Table 4.1.

CLAIM: *The interpretations are well-founded*

ARGUMENT & EVIDENCE: The choice of domains follows from the aircraft level safety analysis and the required choice of interlocks. The *DS*, *DU*, *AS* and *SP* are existing components of the avionics system, with well-known, validated properties. The *Pilot* is trained to follow protocols rigorously.

The customer functional requirements (**Ra** to **Rd**) were provided as an input to the developer team, and were validated with the customer. The development of the safety requirement **RS** is as related in sections A.1 and A.2, and discussed above.

CLAIM: *The overall architecture is feasible*

ARGUMENT & EVIDENCE: The selected overall system architecture has been used successfully on similar safety systems. Inspection confirms that the identified domains and their associated phenomena can interact as needed - aim being to avoid Unshared Information issues (see section 2.4) later in the development.

CLAIM: *Sound judgement followed*

ARGUMENT & EVIDENCE: The development is based on IPDP used successfully on many previous developments and enhanced with POSE to mitigate some known issues with IPDP. Claim this follows normal and not radical design concepts as defined in [131].

CLAIM: *In-air indicator in Aircraft Status System is reliable*

ARGUMENT & EVIDENCE: The in-air indicator is obtained from the weight on wheels and landing gear up indications: if the landing gear is up and there is no weight on the wheels then the aircraft is assumed to be in the air. The landing gear is detected as being up by a number of sensor switches. The switches use a multi-pole arrangement of appropriately selected "Normally open/Normally closed" contacts. This imbues an error detection capability that is used to achieve very good failure rates, well within the required margins.

The PS1: **Initial Problem** structure contains examples of the use of the justification and claims/argument/evidence elements necessary to support the POSE transformation sequence that supports the formation of the initial problem $P_{Initial}$. Just as the justification strength (refer to section 4.1.4) can vary from quite weak (e.g. information based on a data sheet) through to very strong (e.g. diverse evidence from formal proof and test results); so the claim/argument/evidence structure operates similarly. Near the end of a development the evidence to support the final safety case must be strong and compelling. However, at the early stages of a project

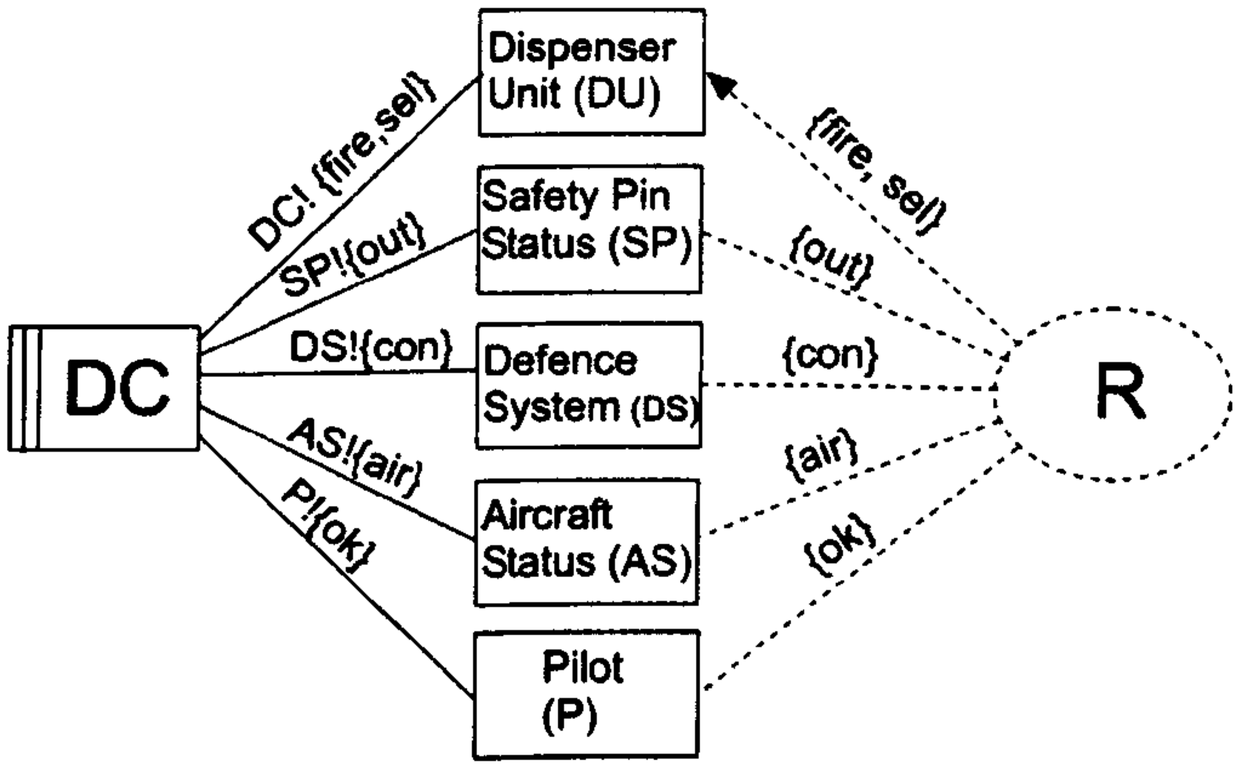

Figure 4.2: PF Problem Diagram for the *DC* Problem, $P_{Initial}$

a "lighter" approach is reasonable where the route to gaining sufficient supporting evidence is clearly outlined, but not as yet necessarily followed. The idea being that the strong argumentation and evidence required by the final safety case will be supplied later in the development. In the early stages the argument strategy is made extant so it can be reviewed to judge if it is capable of supplying the required strength of supporting evidence. This "lighter" structure is consistent with the early development stages where different candidate solution system architectures are evaluated and the most promising candidate selected. Until the analyst has evidence that the candidate architecture is capable of satisfying its safety obligations (i.e. shown it is feasible) it is potentially wasteful to spend effort collecting too much evidence. However, once the candidate system architecture has been shown to be feasible, then the required evidence collection should proceed – the problem transformation step structures such as PS1 above provide an easy means of ascertaining what the current evidence status is and hence what further evidence might be required.

The claim/argument/evidence structures used in PS1, and in the other transformation steps to be introduced (e.g. PS2 below), are all examples of the "lighter" structure in that they outline the strategy but do not (as yet) provide the full level of argumentation and evidence. For example the Claim in PS1 that *The overall*

architecture is feasible is based on similarity with previous successful systems and a review of the domain interactions; so it is expected to be feasible, but the analysis to provide this evidence – based on the PSA results – has not been collected at this point.

4.3 Refining the Solution Architecture

The IPDP Dem/Val phase has detailed architecture analysis as part of its Subsystem Theory of Operation task (refer to section 3.3). This task involves the development of structures and concepts that will enable the requirements to be satisfied. The POSE solution expansion *AStruct* (refer to section 4.1.3) is a good candidate to support this work since it allows various candidate system architectures to be investigated. In this case study, the following *AStruct* encodes the candidate system architecture chosen for the *Decoy Controller*:

$$DecoyContAS[II^{int}_{ok,air,out}, DM^{sel,fire?}_{con}](SC^{fire}_{int,fire?})$$

which includes two extant components, II and DM and one to be found component the safety controller SC. These domains are as defined in section A.4 and in Figure A.2. This architecture can be introduced using the SOLUTION INTERPRETATION and SOLUTION EXPANSION transformations described in section 4.1.3.3. These transformations and the REQUIREMENTS INTERPRETATION transformation can be used to transform the software problem from $P_{Initial}$ into P_{Exp}.

$$P_{Exp}: \quad \begin{array}{l} DS(con)^{con},\ DU(fire,sel)_{fire,sel},\ SP(out)^{out},\ AS(air)^{air}, \\ P(ok)^{ok}, \quad II^{int}_{ok,air,out}, \quad DM^{sel,fire?}_{con}, \quad SC^{fire}_{int,fire?} \quad \vdash \quad R'^{fire,sel}_{con,out,air,ok} \end{array}$$

First the SOLUTION INTERPRETATION transformation is used to introduce the architecture represented by the *AStruct* $DecoyContAS[II, ...DM, ...](SC^{fire}_{int,fire?})$ into the problem, then SOLUTION EXPANSION is used to expand out this *AStruct* and finally REQUIREMENTS INTERPRETATION is used to appropriately modify the requirement from *Rdc* into R' to allow for the introduction of the system architecture.

Experience from using POSE on some avionics examples [87, 86, 85, 88] indicates that this sequence of introducing a candidate system architecture is used frequently, and can be combined into the **PS2 Solution Interpretation and Expansion** problem transformation step represented by the structure shown below.

PS2: ***Solution Interpretation and Expansion*** *applied to problem* $P_{Initial}$ *to form* P_{Exp}

JUSTIFICATION J_2: The identified system architecture, its components and relevant properties are as summarised in section A.4 and in Figure A.2. The justification for this (architecture) solution expansion transformation is that this system architecture was successfully developed as a prototype and is known to be capable of satisfying the functional requirements.

The requirements are transformed from *Rdc* to R' as detailed at the end of section A.4 on page 233 and modified by Table 4.1. However, **R'd** is modified to include reference to the phenomena *int* for representing the interlock status. Note that **RS** is not changed by this transformation. The efficacy of the requirement transformations were validated by independent review. Now R' is:

R'a The *DS* shall command which flare is selected using a field in its *con* message issued to the *DM*. The *DM* shall obtain the selected flare information from this field in the *con* message, and use it in its *sel* message to the *DU* to control the flare selection in the *DU*.

R'b The *DS* shall command the *DM* to issue a *fire*? command in its *con* message.

The *DM* will request the *SC* to send the *fire* command. This shall be the only way in which a flare can be released.

R'c The *SC* shall cause a flare to be released by issuing a *fire* command to the *DU*, which will fire the selected flare.

R'd The *SC* shall only issue a fire command if its interlocks, **represented by** *int*, are satisfied, i.e. aircraft is in air ($air = yes$), *SP* safety pin has been removed ($out = yes$) and *Pilot* has issued an allow a release command ($ok = yes$).

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

P_{Exp} is as defined above. An equivalent PF representation is shown in Figure 4.3 below.

CLAIM: *Rdc and R' are equivalent*

ARGUMENT & EVIDENCE: *Rdc* and R' refer to and constrain the same phenomena, albeit shared between different domains .

CLAIM: *The choice of candidate solution architecture exhibits sound safety engineering judgement*

ARGUMENT & EVIDENCE: The system architecture is chosen to minimise the number and extent of the safety related functions, localising them to simple, distinct blocks in accordance with best practice.

CLAIM: *The chosen solution architecture does not prevent the satisfaction of R' (Feasibility).* **This claim is not yet supported.**

PHENOMENA: The new phenomena introduced by the architecture are:

fire? – Command to release the selected flare type

int – Status of combined interlocks

This POSE system architecture work provides supporting evidence for property **Architecture** from Table 2.3.

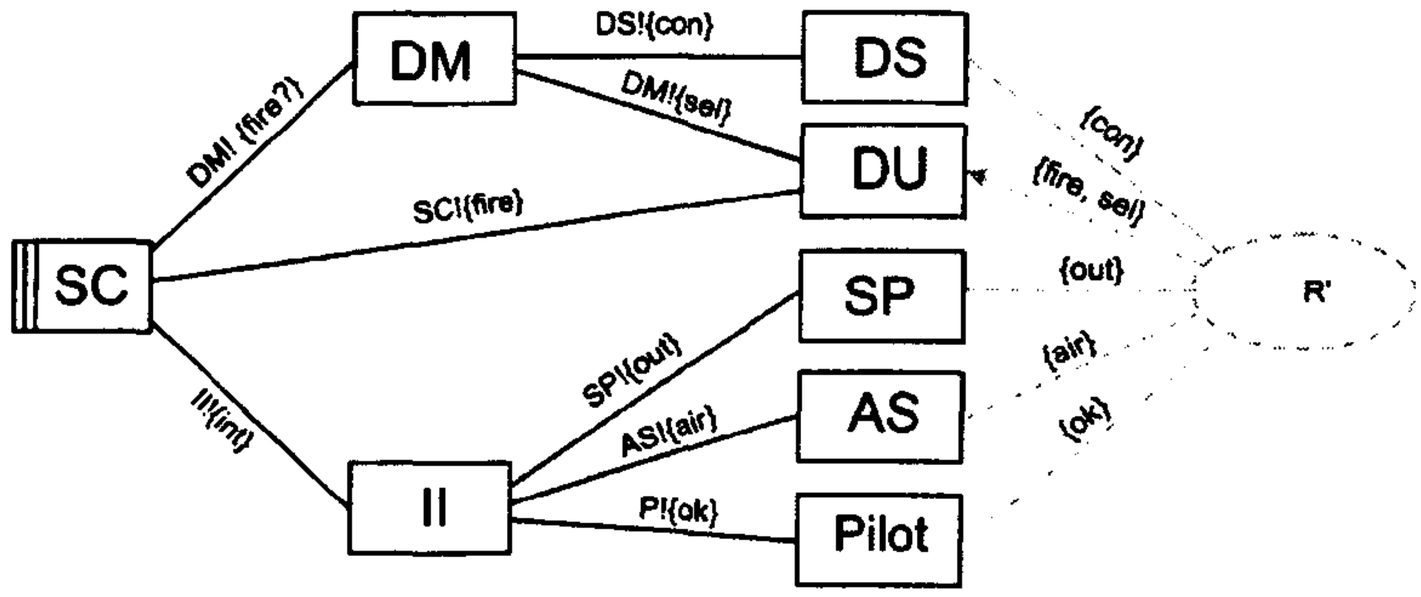

Figure 4.3: Problem P_{Exp} after Solution Interpretation and Expansion

4.4 Preliminary Safety Analysis

The justification of the previous transformation step is incomplete: the *Feasibility concern* remains to be discharged. The related claim is that the chosen system architecture candidate should not prevent an adequately safe solution. In the worst case, to continue the design without checking feasibility uncovers the risk that the final product cannot be argued safe. Traditionally, such risks are mitigated through over-engineering of the solution, but this typically adds to the complexity of the design and increases the development cost.

Here, the risk is managed through a Preliminary Safety Analysis (PSA), applied as part of the IPDP safety analysis tasks in the attempt to discharge the *Feasibility concern*. The goal of a PSA is to:

- confirm the relevance of hazards allocated by the system level hazard analysis;
- identify any further hazards to be added to the list; and
- validate the architecture against the safety targets associated with the identified relevant hazards.

Many techniques can be applied to perform a PSA. In [87] we used a combination of mathematical proof, Functional Failure Analysis (FFA) [109] and functional Fault Tree Analysis (FTA) [130]. The PSA uses the same software problem model as the development, P_{Exp}, which provides supporting evidence for **[SameA1]**.

Note that at this point the PSA is not a POSE transformation *per se* (no POSE schema defines a PSA). Instead it is a technique which we use to discharge one of the claims in the PS2 problem transformation step structure. The continuation of this PS2 structure is shown below.

PS2: ***Solution Interpretation and Expansion*** *applied to problem* $P_{Initial}$ *to form* P_{Exp} *(cont'd)*

CLAIM: *The chosen solution architecture does not prevent the satisfaction of* R' *(Feasibility).* **This claim does not hold.**

ARGUMENT & EVIDENCE: We applied FFA to each architectural component in turn. The results from applying FFA to the *DM* are shown in Table 4.2, where three problem cases were identified: F2, F3 and F5, with 'Yes' in the Hazard column.

Table 4.2: FFA Summary for *Safety Controller*

Id	Failure Mode	Effect	Haz
F1	No *fire*?	Release inhibited	No
F2	*fire*? at wrong time	Inadvertent release	Yes
F3	*fire*? when not required	Inadvertent release	Yes
F4	Intermittent *fire*?	Could inhibit release	No
F5	Continuous *fire*?	Inadvertent release	Yes

A functional FTA applied to *DM* based on the model for P_{Exp} (and Figure 4.3) and using the three FFA problem cases F2, F3 and F5, indicates that a failure in *uP* (systematic or probabilistic) could result in the *fire*? failing on. The *Pilot*'s allow input provides some mitigation, but as soon as this is set ($ok = yes$) a flare will be released, which is undesirable behaviour.

In other words, with this system architecture, H_2 is only protected by the *Pilot*'s allow input. If *fire*? failed on, then as soon as the *Pilot* indicated an intention to allow flare release, by selecting the switch, then the flare would be released, which is not the design intention. Therefore the safety analysis indicates that *fire*? needs to have a safety involved (but not critical) integrity. **The current design does not satisfy the safety requirements.**

The safety analysis indicates that *fire*? needs to have a safety involved integrity. This can only be achieved with the existing design by upgrading all of the design to be safety involved. That is, by assigning *fire*? to the uP, we require that all uP functionality must be at the same safety integrity as *fire*, which is safety involved. However, the bulk of the uP's functionality (timing, BIT, etc.) is not safety-related. Further, any updates to the uP software have to satisfy the safety involved integrity.

The conclusion of the PSA is that the selected DM system architecture is not a suitable basis for the design – no adequate solution can be derived from its parametrisation. Choices at this point include: designing the DM to be safety related, or re-structuring the DM system architecture to partition the safety and non-safety elements. The first option is undesirable due to the expense and long term impact, i.e. timing and selection are not safety functions and are expected to be fine-tuned to support different flare types. Making this safety-related would have a detrimental impact on the affordability of the solution. The second option is more appealing, and was the option chosen for this development. This case study is continued in Chapter 5.

The fact that PSA could be applied to the POSE problem model, P_{Exp}, provides some support for the property **Right Abstraction**, although more evidence is required to support this fully.

4.5 Discussion

From our initial work on using POSE on the *DC* case study we can now start to assess the ability of POSE to support the IPDP and address the properties and issues we have identified.

4.5.1 Properties and IPDP Issues

The work reported in section 4.1.3 showed that context interpretation was used to develop a good understanding of the environment context to identify the domains involved and investigate their interfaces and communication mechanisms. The practical result of following this process in IPDP was a thorough understanding of the environment context, which provides mitigation for **[EnvirI1]**.

The work reported in section 4.1.3 also demonstrated that the effort in applying the requirements interpretation addressed the concerns raised by **[RequI2]** in comparison with the real development summarised in Appendix A. It showed that the requirements produced using this process were more focussed and had less contradictions and consistency issues, when compared with performing the same tasks without POSE. It made explicit the relationship between the requirements and the environment domain phenomena, which assisted the later modelling work.

The PSA work of section 4.4 demonstrated that POSE has potential with respect to supporting the IPDP safety analysis tasks by providing a suitable model structure on which the analysis could be based. The PSA used the model based on P_{Exp} and Figure 4.3, along with the associated domain, phenomena and behaviour descriptions referenced from the POSE steps *PS1* and *PS2*. That is, the safety analysis used the same model as the development (P_{Exp}) in agreement with **[SameA1]**. As expected, POSE (a PO approach) is very good at addressing **[EnvirI1]** and **[RequI2]**. However, this POSE model structure is available much earlier in the development life cycle than the Preliminary Design phase. In fact the model could be

formed at the end of the Dem/Val phase and the feasibility check could be used as a checkpoint in the IPDP SDR. This is a very significant factor because if it can be demonstrated in a more general context, then POSE embedded within IPDP would allow a meaningful safety analysis to be conducted much earlier in the IPDP life cycle than is currently the case - thus satisfying **[LateI3]** identified in section 3.4.5.

At the Dem/Val SDR phase the design is more abstract and compact than at the end of Preliminary Design, and it is much easier to incorporate changes. It follows that any iteration back associated with failing the PSA feasibility check conducted at the end of the Dem/Val phase can be contained within a relatively small number of related IPDP tasks within the phase. In contrast, failing the feasibility check conducted during EMD Preliminary Design involves iterating back over many tasks over a number of phases. Therefore, checking at the Dem/Val phase allows the necessary corrective action to be instigated earlier and have less impact on cost and schedule compared with finding a problem later in the life cycle during Preliminary Design. The inference is that performing the PSA early would address **[LateI3]** and also cover the part of **[TraceI4]** concerned with iteration.

In the *DC* case study the initial system architecture failed the feasibility check and this showed that there is a need to provide direct support for iteration in the life cycle, and it supported the findings of a significant number of other IPDP projects that required re-working. Basically a decision point is required such that if the architecture model passes the feasibility check then the project can continue to the next phase of the development, otherwise there is a need to iterate back within the current phase to allow corrective action to be applied. As noted above, performing the check during SDR has the advantage that the iteration can be incorporated into a small number of related tasks thus minimising its detrimental impact. Integrated as part of IPDP, iteration then is part of the normal, expected task transactions and does not attract the adverse senior management scrutiny that currently occurs with the existing non-POSE, ad hoc iteration handling mechanisms. The need for

a decision point and how to achieve support for iteration will be considered further in Chapter 6.

Inspection of the model used for the safety analysis (see P_{Exp} and Figure 4.3) indicates that it is more complex than necessary and involved domains that were not directly relevant to the issue under investigation. A more efficient safety analysis would be obtained if it were possible to remove these extraneous domains and just concentrate on the core domains of the safety problem. Another driver for removing extraneous domains is the fact that the requirement R' is still expressed in terms of the system environment, and not in terms of the system to be designed. For example, in the case study **R'd** involves terms in *air* = *yes*, yet the *SC* has no interface to the *AS*, so as they stand the requirements R' are not implementable with respect to the *SC*. This means that there is no direct way of extracting the specification of *SC* using the problem P_{Exp} and thus at this point **[NecessaryP]** has not been demonstrated. To achieve the goal of extracting the specification, it is necessary to be able to transform R' so that it more directly relates to the *SC*. The next chapter introduces the POSE problem progression transformation (PPT) as a means of mitigating these two issues by removing extraneous domains whilst correctly re-writing the requirements to compensate.

The adequate completion of IPDP task activities was identified as an important factor in the successful application of IPDP and an area where problems have been detected in the past, this was captured in section 3.4.5 as **[IncomI5]**. The work in section 4.2 suggests that POSE can assist with this completeness issue in three main ways:

- the framework ensures that the environment domains are adequately considered, requirements assimilated and also ensures that the distinction between the indicative and optative properties are well understood;

- the POSE transformations provide a mechanism for completing the objectives

of certain important tasks in the development;

- the POSE problem transformation steps (see section 4.1.4.2) provide a template structure for ensuring that all the relevant factors in completing a development task have been addressed and this can be easily reviewed and validated.

However, more work is required to ensure that all the necessary task activities have been identified and accurately captured. Finally, POSE is known to have the capability of supporting formal development processes [39, 88], but evidence to support this has not been provided as yet.

4.5.2 IPDP Process

These initial results are encouraging and indicate that POSE can be used as part of the IPDP. Further, the *DC* case study work shows that the early IPDP phases can benefit from the structuring of the requirements that this can bring. In particular, inspection of the POSE capabilities and the safety work presented thus far in section 4.4 indicate that it would be useful and beneficial to develop a front-end process based on the POSE transformations targeted at embedded real-time safety systems (which are the major focus of this work), since there is good evidence from the *DC* case study that POSE addresses **[SameA1]**, **[EnvirI1]** and **[RequI2]**. However a number of improvements to better achieve these goals (as discussed above in section 4.5.1) pertaining to derivation of the specification S, early safety analysis, improved iteration, safety case support, model simplification, requirements transformations, the completion of all the necessary activities and support for formal development processes were identified as needing attention. These will be developed further in the next few chapters.

The use of POSE within IPDP, and a measure of how well POSE integrates with IPDP, can be obtained from inspection of Table 4.3, where the "Ref." entries refer to sections in this document. The table shows that POSE works well with IPDP for

this single case study, but at this point there is not enough evidence to fully support the properties [**IntegA3**] and **Integrates**.

IPDP Phase	IPDP Sub-Phase	IPDP Ref.	POSE Artefact	POSE Ref.
CE	System Context	3.2.1	Initial Context	**4.1.2.1**
CE	System Context	3.2.1	Context Int.	**4.1.3.1**
CE	Initial Fun. Req.	3.2.2	Requirement Int.	**4.1.3.2**
Dem/Val	Develop Fun. Req.	3.3.1	Requirement Int.	**4.1.3.2**
Dem/Val	Sys. Theory of Op.	3.3.1	AStruct – DCS[](DC)	**4.1.3.3**
Dem/Val	Sys. Theory of Op.	3.3.1	$P_{Initial}$	**4.1.3.3**
Dem/Val	Sub-Sys. Theory of Op.	3.3.2	P_{Exp}	**4.3**
Dem/Val	Safety Analysis	3.3.4	-	-
EMD	Safety Analysis	3.4.3	PSA	**4.4**

Table 4.3: Evidence to Support POSE Properties

In summary, the *DC* case study work provides good evidence that using POSE within IPDP will allow [**SameA1**], [**EnvirI1**], and [**RequI2**] to be addressed, and provides some evidence to support [**IntegA3**]. In addition, it provides some supporting evidence for the **Context**, **Architecture**, **Avoid Bias**, **Model Reality**, **Validation**, **Simplification** and **Integrates** properties identified in Chapter 2, and to some extent property **Right Abstraction** too. The problem transformation steps extension introduced in section 4.1.4.2 assists with the applicability of POSE by providing a convenient mechanism for grouping related tasks together into a single entity; e.g. the PS1 structure in section 4.2.

We have summarised the evidence in Table 4.4, where "-" means no evidence has been provided by this work to support the property/issue, "e" means that some evidence has been provided by this work to support the property/issue, "ee" means that good, corroborated evidence (e.g. from two case studies) has been provided, and "E" means that significant corroborated evidence (e.g. multiple case studies) has been provided by this work. Entries shown in bold indicate that evidence is obtained from the results of this chapter. The table also includes references to the external papers, under "Previous Work", where evidence is provided that POSE

supports the property or mitigates the issue.

Property/Issue	Previous Work	**Chapter 4**
Context	[41, 87]	**e**
Architecture	[41, 87]	**e**
Spec. Join	[39, 83]	-
Avoid Bias	[39]	**e**
Model Reality	-	**e**
Validation	-	**e**
Simplification	[41, 87]	**e**
Tool Support	[38, 87]	-
Right Abstraction	[41, 87]	**e**
Integrates	[88]	**e**
NecessaryP	[41, 83]	-
SameA1	[88]	**e**
FormalA2	[41, 88]	-
IntegA3	-	**e**
EnvirI1	[87, 88]	**e**
RequI2	[87, 88]	**e**
LateI3	[85, 88]	-
TraceI4	[85, 88]	-
IncomI5	-	-
ValidI6	[88]	-

Table 4.4: Evidence Status for POSE after Chapter 4

4.6 Chapter Summary

In this chapter we have introduced POSE and evaluated it within IPDP based on the *DC* case study. The outcomes of this exercise were discussed in section 4.5.2. The latter concluded that POSE within IPDP addressed **[SameA1]**, **[EnvirI1]** and **[RequI2]**, and that it could address the remaining issues with some development and extensions - these are the subject of the next two chapters.

The contribution of this thesis so far to the knowledge concerning POSE and its use with respect to supporting these properties and mitigating the six issues is summarised in Table 4.5, where "vanilla" POSE is as defined in [41, 39, 40], i.e., not an expansion introduced by this work. The "Reference" column refers to section

numbers unless otherwise stated.

Property	Contribution Description	Reference
All of Table 2.3	Identified a set of properties appropriate for SSSD.	2.5 Table 2.3
[NecessaryP]	Identified necessary property for deriving specification, S.	3.4.4
[SameA1] [FormalA2] [IntegA3]	Identified three useful properties that the SSSD approach should support.	3.4.4
[EnvirI1]	Identified IPDP issue for mitigation	3.2.1, 3.3.1 & 3.3.4
[RequI2]	Identified IPDP issue for mitigation	3.2.2, 3.3.3 & 3.4.1
[LateI3]	Identified IPDP issue for mitigation	3.2.3, 3.4.3, 3.3.3, 3.3.4
[TraceI4]	Identified IPDP issue for mitigation	3.2.3, 3.3.2, 3.4.2
[IncomI5]	Identified IPDP issue for mitigation	3.2.4, 3.3.1
[ValidI6]	Identified IPDP issue for mitigation	3.3.5
[EnvirI1]	*DC* case study demonstrated "vanilla" POSE mitigates this issue.	4.1.3 4.5.1
[RequI2]	*DC* case study demonstrated "vanilla" POSE mitigates this issue.	4.1.3 4.5.1
[SameA1]	*DC* case study demonstrated "vanilla" POSE supports property.	4.4 4.5.1
[IntegA3]	*DC* case study demonstrated "vanilla" POSE provides some support.	Table 4.3 4.5.2
Context	*DC* case study demonstrated "vanilla" POSE provides some support.	4.1.2 & 4.1.3
Avoid Bias	*DC* case study demonstrated "vanilla" POSE provides some support.	4.1.2 & 4.1.3
Model Reality	*DC* case study demonstrated "vanilla" POSE provides some support.	4.1.2 & 4.1.3
Architecture	*DC* case study demonstrated "vanilla" POSE provides some support.	4.3
Validation	*DC* case study demonstrated "vanilla" POSE provides some support.	4.1.2
Simplification	*DC* case study demonstrated "vanilla" POSE provides some support.	4.1.3
Right Abstraction	*DC* case study demonstrated"vanilla" POSE provides some support.	4.4
Integrates	*DC* case study demonstrated "vanilla" POSE provides some support.	4.5.2 & Table 4.3

Table 4.5: Thesis Contribution Summary after Chapter 4

Chapter 5

Enhancing the Process Using POSE

Chapter 5 specifically investigates the properties **[NecessaryP]**, **[FormalA2]**, **Validation**, **Simplification** and the issues **[TraceI4]**, **[LateI3]**, **[ValidI6]**; and it discusses the evidence collected to support them and the other properties through the use of representative case study examples. The chapter concludes by considering this supporting evidence in the status table, first introduced in section 4.6.

5.1 From Problems to Software Solutions

Jackson's work [53] distinguishes between software requirement and specifications. Simply stakeholders should understand the requirement as constraining their context, whereas the specification should be implementable [133]. The requirement should be grounded in the reality of the context for which the machine is to be designed [133], whereas the specification should be defined solely in terms of the phenomena at the interface between the machine and its context.

Other approaches, particularly formal ones [43, 111], recognise the gap between specification and code and provide techniques to bridge it. POSE with its generalised

notion of solution addresses the larger gaps (from requirements in context through the specification to code).

Within Problem Frames a number of authors have suggested approaches to bridge the gap between requirement and specification. Jackson himself in [53] sketches out the idea of problem progression. Starting from this idea, Li [74, 75], expands and completes the idea of problem progression exploiting the concept of causality between phenomena within a graph theoretical framework. Other work by Seater and D. Jackson [113, 114] propose a notion of progression based on the addition of domain assumptions.

In POSE too, there is a notion of progression which is based on the combination of two transformation schemas [36, 39, 40, 41]. This is very general in nature and provides no particular guidance or heuristics for its application. The work in this section will address how to make it practical in the context of safety critical engineering by adopting a similar notion of causality as Li, and demonstrating how this integrates with the theoretical underpinnings provided by POSE.

5.1.1 Problem Progression

In [40], Hall, Rapanotti and Jackson defined transformations from which a POSE notion of problem progression can be derived. Their transformations are perfectly general and assume nothing of a domain, its phenomena, the relationships between phenomena or how the requirements refers to or constrains the phenomena. As a consequence the transformation wrought on the requirement can make no use of domain, phenomena or requirements structure. For embedded systems an assumption of causality between phenomena allows a more grounded engineering view of problem progression. The assumption of causality used in this work is based on the approach outlined in [94]. Our engineering form of problem progression is related to Li's work [74, 75], which is another general theoretical framework but this time based

on the assumption of causality between phenomena within domains. One benefit of Li's approach is that it shows precisely how to construct a new requirement from the old one. However, Li's work applies only in Problem Frames.

5.1.1.1 Problem Progression in Problem Frames

The goal of a problem progression sequence can be explained graphically with reference to the Problem Frame problem progression diagram shown in Figure 5.1. This is an elaboration of the progression shown on Page 103 of [53] and that used by Li [74] (Note that this is only an illustration and not the generic case. Li provides a complete taxonomy in [74]).

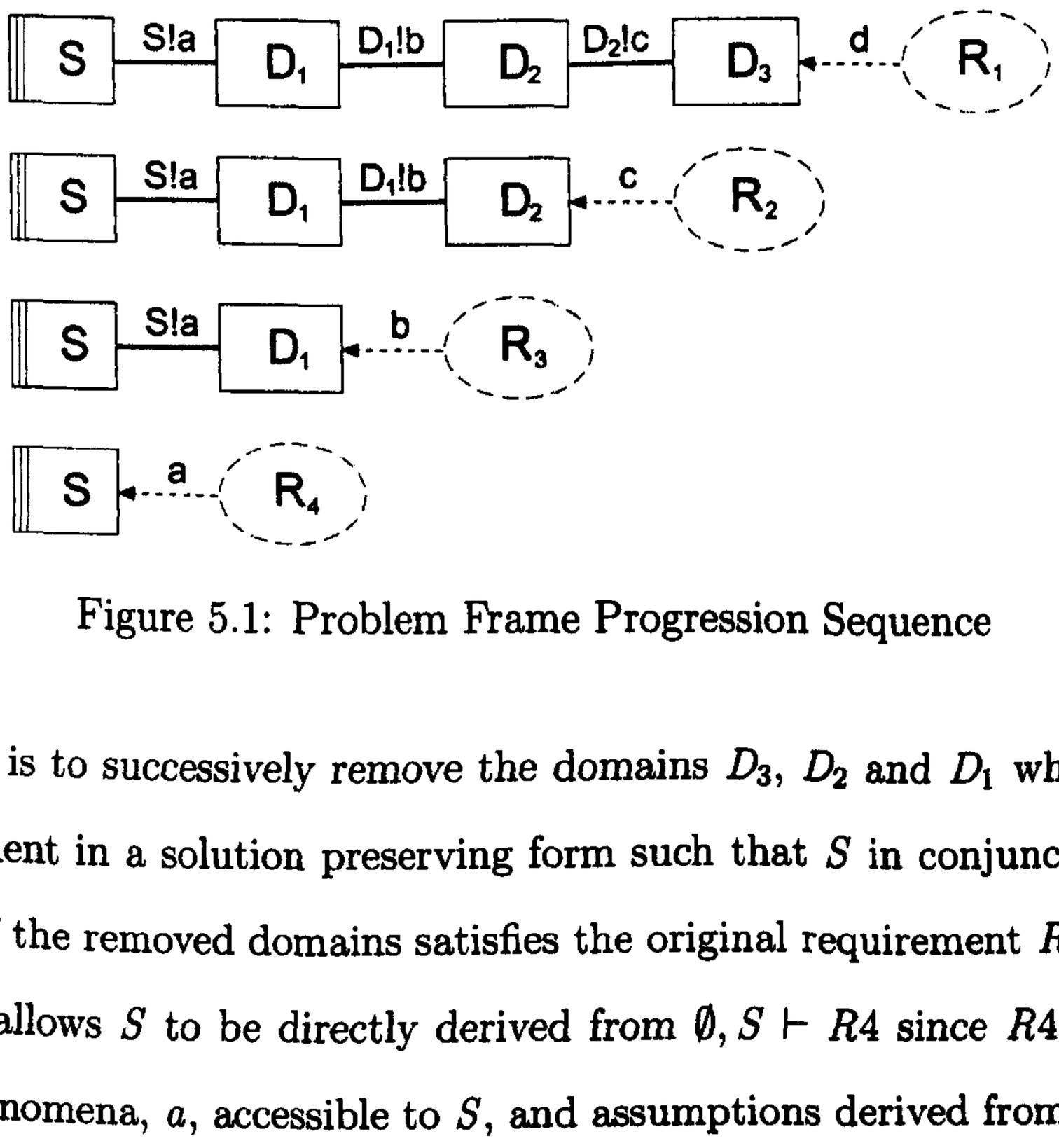

Figure 5.1: Problem Frame Progression Sequence

The goal is to successively remove the domains D_3, D_2 and D_1 whilst rewriting the requirement in a solution preserving form such that S in conjunction with the properties of the removed domains satisfies the original requirement R_1. Doing this progression allows S to be directly derived from $\emptyset, S \vdash R4$ since $R4$ is written in terms of phenomena, a, accessible to S, and assumptions derived from the removed domains.

By combining the work of Jackson and Li, progression in Figure 5.1 can be described as follows. In the initial problem, the requirement, R_1, is expressed in terms of phenomena in the real world – represented by d. Following the notion

of causality used by Li [74], assume that there exists a causal relation between phenomena c and d induced by the properties of the the domain D_3, such that if c occurs then d occurs: we call such a relation the *useful behaviour* relation and indicate such a relation as $c\ D_3\ d$. Then we can omit D_3 and define a new requirement, R_2, expressed in terms of phenomenon c and derived from R_1 with the addition of assumptions capturing the useful behaviour relation $c\ D_3\ d$. In doing so, the original problem has been progressed to a new problem with a simplified context, in such a way that a solution which satisfies R_2 in the progressed problem, will also satisfy R_1 in the original problem.

In practice the engineering domains used in the architecture could have a variety of possible behaviours, but we are only interested in that part of the domain's behaviour that corresponds to the relation $c\ D_3\ d$ – hence it is called the useful behaviour relation. It is also worth noting that this useful behaviour relation corresponds to the *Environment Constraint* introduced in section 2.4.

When progression is based on causality it is important to define relations such as $c\ D_3\ d$ properly. Li provides a basic taxonomy [74] of relevant types of causal relations. Among them is the notion of *biddable* causality which relates to human behaviour. While in general human behaviour is unpredictable, there are cases in which people can be assumed to follow particular instructions or orders, from which causal relations can also be assumed. This type of causal relation is relevant to our work, as the humans involved in operating the systems produced by the Company tend to be highly trained pilots or other skilled military personnel, so that an approximation of causal behaviour can be claimed for benign scenarios.

5.1.1.2 Problem Progression in POSE

In POSE the domain to be removed is termed the *to-be-progressed domain* and is removed using a two step process as follows [39]:

- first the phenomena associated with the to-be-progressed domain are "un-

shared" by internalising them into it, making an assumption in the requirement equivalent to the constraints that the to-be-progressed domain places on its shared phenomena – this is the SHARING REMOVAL transformation shown below

- second a domain that shares no phenomena may be removed from the context of a problem – this is the DOMAIN REMOVAL transformation shown below.

The two rules are as follows:

$$\frac{W, D(c,o) : Desc_D, S \vdash R' : F'}{W, D_o^c : Desc_D, S \vdash R : F} \text{ [SHARING REMOVAL]}$$

where the requirement becomes $R' = R_{ref,o}^{con,c}$ and $F' = c\ D\ o\ , F$. The relation cDo is the useful behaviour relation introduced in section 5.1.1.

$$\frac{W, S \vdash R}{W, D_\emptyset^\emptyset, S \vdash R} \text{ [DOMAIN REMOVAL]}$$

The key point is that constraints that the to-be-progressed domain places on its shared phenomena (the useful behaviour of the domain) are captured within the requirement as an assumption. The requirement before the progression is likely to refer to phenomena that have been removed as part of the progression, therefore a REQUIREMENTS INTERPRETATION transformation is usually necessary to rephrase the requirement in terms of the phenomena extant in the progressed problem description. As noted above, problem progression in POSE is quite general – it can be applied to any domain in the problem context – and there remain difficulties in applying it associated with identifying the appropriate part of the requirement and how it should be refined. The approach adopted in this work is to restrict the application of the progression rules in a way which facilitates the analysis and this is considered in the next section.

5.1.2 POSE Problem Progression for Safety Systems

The approach used to derive the problem progression transformation sequence (PPT) developed in this work is based on the general POSE problem progression introduced above, but it is made more specific to serve the needs of practical engineering. In particular, we apply progression under the following conditions: that causal relations among phenomena are identified and used to establish assumptions in the derived requirements; that only domains whose phenomena are directly referred to or constrained by the requirement are the subject of progression.

We call such domains *adjacent* to the requirement, as in Problem Frame notation they would be directly linked to the requirement by a dashed line (if referred to) or a dashed arrow (if constrained).

With reference to Figure 5.1, let us assume a causal relation between phenomena c and d, whose derived assumption we denote by $Assumption(c, d)$. The first step of progression in the figure corresponds to the following POSE transformations:

$$\dfrac{\dfrac{\dfrac{D^b_{1a},\ D^c_{2b},\ S^a \vdash R^c_2}{D^b_{1a},\ D^c_{2b},\ S^a \vdash R^d_1 \wedge Assumption(c,d)}}{D^b_{1a},\ D^c_{2b},\ D^{\emptyset}_{3\emptyset}(c,d),\ S^a \vdash R^d_1 \wedge Assumption(c,d)}}{D^b_{1a},\ D^c_{2b},\ D^d_{3c}\ S^a \vdash R^d_1}$$

$[RequirementInt.] \ll J_1 \gg$
$[DomainRemoval]$
$[SharingRemoval]$

based on the properties captured by $Assumption(c, d)$. In other words, J_1 has to justify that R^c_2 is in some sense equivalent to R^d_1 through $Assumption(c, d)$.

Repeated application of the same combination of POSE rules to the problems in Figure 5.1 will remove domain D_2 and then domain D_1. Each requirement interpretation step will need justification, namely that

R^b_3 is in some logical sense equivalent to $R^c_2 \wedge Assumption(b, c)$

and that

R^a_4 is in some logical sense equivalent to $R^b_3 \wedge Assumption(a, b)$

This combination of SHARING REMOVAL, DOMAIN REMOVAL and REQUIREMENTS INTERPRETATION transformations is called the PPT in this work and in the sequel a *problem transformation step* will be used to control each application of the PPT, as demonstrated in the *DC* case study problem progressions considered in section 5.1.3.

The PPT is based on domain properties and causal relations between phenomena. As we will see, it is amenable both to informal and formal logical analysis: the latter is possible when both requirements and assumptions can be expressed logically and their equivalence proved through logical implication. In section 5.1.3 we illustrate an informal PPT on the *DC* case study. Later case studies will illustrate a formal PPT based on the use of Alloy [52].

The PPT approach has the advantage that it is easy to apply since it uses knowledge about the domains and the known requirement to guide the refinement of the requirement; its disadvantage is that only in the last validation step can you demonstrate that the derived specification S satisfies the requirement. In theory it is possible to go through a derivation sequence and derive a specification that does not fully satisfy the requirement, although this did not occur in the case studies reported in this work.

In summary, the PPT uses problem knowledge to guide its application and to refine the requirement as domains are removed. The problem progression can be seen as a form of problem simplification since it allows the specification to be derived by simplifying the problem (by removing domains), thus it provides support for property **Simplification** identified in section 2.5. The PPT application also validates that the specification, S, you derive from applying the PPT is capable of satisfying the original requirement and thus supports the issue **[ValidI6]** and the **Validation** property, as well as property **[NecessaryP]**.

The assumption of causality among phenomena is an important aspect of the PPT approach which is reasonable for the embedded real time safety systems that

are the focus of this work. A domain is often representative of a piece of avionics equipment that has a particular function – give it the right input and it will produce the desired output. As part of the initial design, these domains are arranged in a particular system architecture to produce the required behaviour.

5.1.3 Problem Progression Applied to the *DC*

An example of applying the PPT process to remove domains will now be presented based on the *DC* case study first introduced in Chapter 4. The model of the *DC* problem P_{Exp} represented in Figure 4.3 is quite complex and the requirement phenomena do not correspond to the *SC* Machine phenomena, i.e., the current requirement, R', is not feasible with respect to the Machine, *SC* since it cannot be implemented with the information available. This is resolved by applying the PPT to remove domains and rewrite the requirement accordingly so that the Machine specification can be derived, and then validated that it satisfies R'. As noted in section 5.1.2, the problem progression begins with domains adjacent to the requirement. Inspection of Figure 4.3 indicates that the *Pilot* switch selection domain is a suitable first candidate for removal. The removal proceeds by following the process introduced in section 5.1.2. These steps are captured in the *Pilot Domain Removal* problem transformation step PS3 shown below.

PS3: Application of the PPT for Pilot Domain Removal to problem P_{Exp} to form P_{Red1}

INCLUDES: None

JUSTIFICATION *J*: The *Pilot* domain removal is based on the process given in section 5.1.2. The *Pilot* domain is adjacent to the requirement R'. The *Pilot* domain control phenomena *ok* maps directly to its output behaviour phenomena which is also *ok*. Therefore *ok* will appear in the reference part of the requirement phenomena both

before and after the transformation. The requirements R' are as detailed in problem transformation step PS2 on page 95. Inspection of R' indicates that only **R'd** involves a reference to phenomena *ok* – now consider the transformation from **R'd** into **R1d**:

$R'd$: Pilot has issued an allow release command ($ok = yes$).

$R1d$: II observes pilot input of $ok = yes$.

Pilot Domain Description: The pilot sets the Allow switch to active to indicate that the intention is to allow flare release, otherwise it is set inactive. The Allow switch uses a multi-pole arrangement of appropriately selected "Normally open/Normally closed" contacts. This imbues an error detection capability that is used to achieve very good failure rates, well within the required margins. The setting of the Allow switch is determined by the *Pilot* domain and is output using the phenomena *ok*, where $ok = yes$ means the Allow switch is active, whilst $ok = no$ means the allow switch is inactive. Therefore $ok = yes$ corresponds to allow flare release, whilst $ok = no$ corresponds to inhibit flare release.

The justification for this problem progression is that $R1d \wedge Assumption(ok, ok) \Rightarrow R'd$.

The $A_1 = Assumption(ok, ok)$ is given by the *Pilot Domain Description*. Inspection of $R1d$ in conjunction with the *Pilot Domain Description* shows they imply $R'd$ as required.

This requirement transformation does not require additional information not already in the interface nor does it refer to future events. However it does rely on the useful behaviour from the domain being removed being "saved" as part of the assumption that justifies the transformation – it is an example of an Environment Constraint non-feasible requirement which is discussed in a POSE context in [83]. As noted above, this useful behaviour contribution is contained in the *Pilot Domain Description*.

The PPT application transforms the problem from P_{Exp} into P_{Red1}, and R' is transformed into $R1$ as follows:

R1a The *DS* shall command which flare is selected using a field in its *con* message issued to the *DM*. The *DM* shall obtain the selected flare information from this

field in the *con* message, and use it in its *sel* message to the *DU* to control the flare selection in the *DU*.

R1b The *DS* shall command the *DM* to issue a *fire*? command in its *con* message. The *DM* will request the *SC* to send the *fire* command. This shall be the only way in which a flare can be released.

R1c The *SC* shall cause a flare to be released by issuing a *fire* command to the *DU*, which will fire the selected flare.

R1d The *SC* shall only issue a fire command if its interlocks, represented by *int*, are satisfied, i.e. aircraft is in air ($air = yes$), *SP* safety pin has been removed ($out = yes$) and *II* **observes pilot input of** $ok = yes$.

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

The domain *Pilot* is no longer referenced in the model and can be removed.

The software problem is transformed to P_{Red1}:

$$P_{Red1}: \quad \begin{array}{l} DS(con)^{con},\ DU(fire, sel)_{fire,sel},\ SP(out)^{out}, \\ AS(air)^{air},\ II(ok)^{int}_{ok,air,out},\ DM^{sel,fire?}_{con},\ SC^{fire}_{int,fire?} \end{array} \vdash R1^{fire,sel}_{con,out,air,ok} \wedge A_1$$

CONCERNS: The control input phenomena *ok* is the same as the domain behaviour output phenomena for this reference-type problem progression. This can be a symptom of over-specification.

CLAIM: *The pilot's intention with respect to the allowing of flare release is represented by the phenomena ok, so no over-specification requirement issue arises.*

ARGUMENT & EVIDENCE: This is covered by the *Pilot Domain Description* and the validation as described above.

The first application of the PPT has removed the *Pilot* domain as detailed in

the problem transformation step PS3 above. The removal of the domains *AS*, *SP* and *DS* are detailed in the problem transformation steps represented by PS4, PS5 and PS6 respectively that are contained in Appendix B.

After four applications of the PPT to remove the *Pilot*, *AS*, *SP* and *DS* domains the requirement is transformed into *R*4:

R4a The *DM* shall decide which flare to select by observing a field in the *con* message it receives. The *DM* shall obtain the selected flare information from this field in the *con* message, and use it in its *sel* message to the *DU* to control the flare selection in the *DU*.

R4b The *DM* shall observe the *con* message it receives and issue a *fire*? command if commanded. The *DM* will request the *SC* to send the *fire* command. This shall be the only way in which a flare can be released.

R4c The *SC* shall cause a flare to be released by issuing a *fire* command to the *DU*, which will fire the selected flare.

R4d The *SC* shall only issue a fire command if its interlocks, represented by *int*, are satisfied, i.e. *II* observes input $air = yes$, *II* observes input $out = yes$ and *II* observes pilot input of $ok = yes$.

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

and the software problem is transformed into P_{red}:

$$P_{Red}: \quad \begin{array}{l} DU(fire, sel)_{fire,sel}, \quad II(ok)^{int}_{ok,air,out}, \\ DM^{sel,fire?}_{con}, \quad SC^{fire}_{int,fire?} \quad \vdash \quad R4^{fire,sel}_{con,out,air,ok} \wedge Assump \end{array}$$

where *Assump*. is the conjugated collection of assumptions from each PPT application.

In PF problem diagram form, P_{Red} is as shown in Figure 5.2. At this point the remaining domains are all adjacent to the SC and also adjacent to the requirement $R4$. Due to the detailed (over-specified) nature of the requirement used in this example, the requirement phenomena associated with $R4$ relate directly to the domain phenomena visible by the SC. Therefore, the specification of the SC can now be derived directly using the P_{Red} entailment and thus satisfying property **[NecessaryP]**. Note that by applying a further REQUIREMENT INTERPRETATION transformation the requirements phenomena (*fire*, *fire*?, *sel*, *air*, *ok*, *out*) can be directly related to their corresponding domain phenomena (*int*, *fire*, *fire*?, *sel*) through the requirement $R4$.

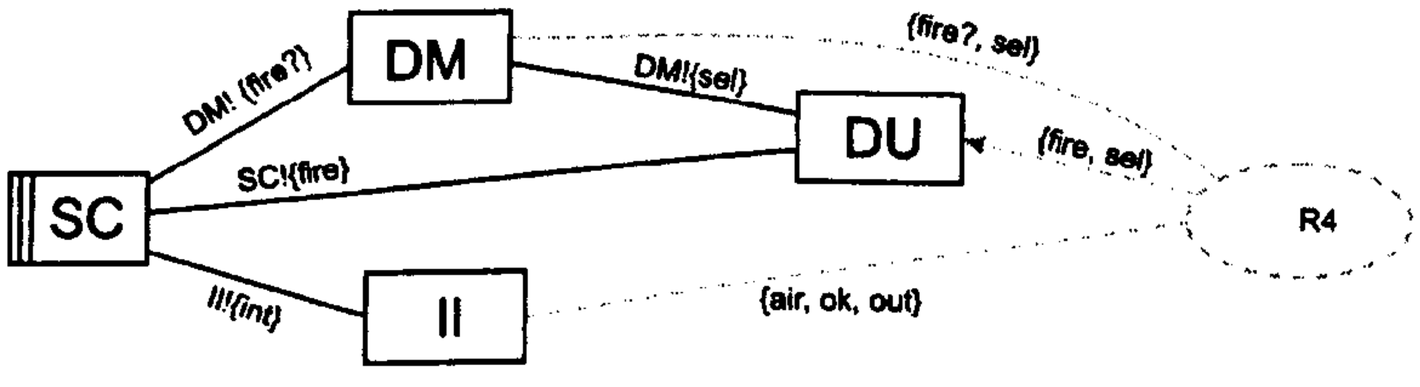

Figure 5.2: *DC* Problem after PPT, P_{Red}

A problem with the *DC* case study development is that the *effect* of a domain and the causal chain phenomena associated with this *effect* were not clearly distinguished. For example, when applying the PPT to remove the *Pilot* domain (refer to problem transformation step PS3 on page 115) the causal chain control phenomena *ok* appears in the before and after requirement statements. The requirement **R'd** contains the clause "*Pilot* has issued an allow a release command ($ok = yes$)", which indicates that $ok = yes$ corresponds to the pilot's intention to allow flare release. However, the latter fact was established from the *Pilot Domain Description* and is not really pertinent to the requirements at this point. In reality, the $ok = yes$ term was added because the specifier knew the system well and was tempted to add too much detail and thus over-specify. Similar issues surround the other domains.

The specification could be derived after step PS6 because the requirement is more detailed than necessary. The over-specification is indicated by the fact that

the requirement refers to phenomena and domains that are not adjacent to it, but are embedded in the problem (e.g., $R'b$ from section 4.3 refers to *fire*? and DM). A useful check that can be derived from this is that a requirement is at an appropriate level of abstraction if it refers to domains and phenomena that are adjacent to it – if it refers to domains or phenomena that are embedded in the problem then it is too detailed. This relates to the **Right Abstraction** property identified in section 2.5.

5.2 Safety Analysis Improvement [LateI3]

The earlier PSA work reported in section 4.4 made use of a combination of FFA to identify hazards and functional FTA to investigate them. The FFA was applied to the main system functions and the results obtained from this analysis were found to be adequate, but the process used did not make full use of the capabilities of the model derived from the POSE software problem description. Specifically the model explicitly identifies the domain phenomena and how they inter-relate, and this information is not directly used in the FFA. Further, FFA has a number of strengths and weaknesses. Its main strength is that it can be readily applied to many situations. Its major weakness is that it is only likely to be 80% effective at detecting the issues of interest [90]. This suggests that it should be used in conjunction with other, diverse techniques such that the combination is likely to be more effective than just FFA alone. The availability of the phenomena information indicates that a HAZOPS style [6, 106] of analysis based on applying the guide-words to the phenomena might be more apposite for this analysis task. This section will consider how FFA, HAZOPS and FTA can be used in combination to achieve an effective safety analysis process to support the PSA; it will then use this safety analysis process on the PSA re-applied to the DC case study. It will also consider how safety analysis can be used as part of a formal development. The goal is to ensure that the safety analysis is effective in support of the issue **[LateI3]**.

5.2.1 Improving the Preliminary Safety Analysis Process

The aim of this section is to consider how the model formed by the POSE software problem description can be used to support the PSA task. First the model will be used to improve the PSA on a non-formal development, and then the use of the model to support the PSA on a formal development using Alloy will be considered.

5.2.1.1 Improving a Non-Formal PSA

Consider the possible hazardous failure modes in a domain D that relates to another domain SC through the phenomena:

a : controlled by SC, observed by D

b : controlled by D, observed by SC.

These entities are shown in Figure 5.3 along with the failure space of the domain D. The failure space of domain D is split into four distinct regions:

1. *D fails but not Hazardous*
2. *D fails Hazardous not related to a*
3. *D fails Hazardous related to a*
4. *D fails & this causes b issue*

The failures in the first region are not hazardous, so these failures are not considered further. The failures in the second region are hazardous and are caused by internal phenomena in domain D. These failures can be detected by applying an FFA to domain D's functions. However, this identifies the hazard but not which phenomena caused it. Understanding the causality can be important when trying to identify an appropriate mitigation strategy, and FFA on its own does not provide this information.

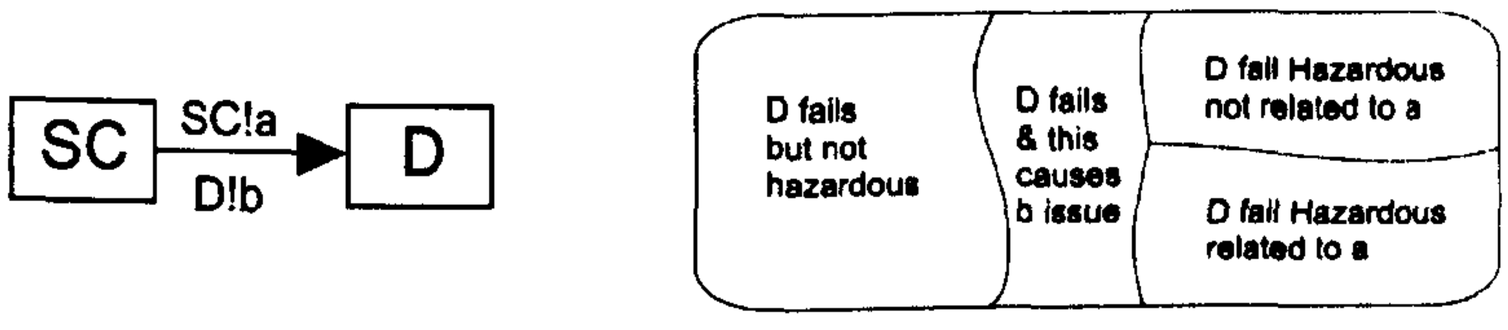

Figure 5.3: Context Diagram and Failure Space for Domain D

The failures in the third and fourth regions involve the relationship between the domain D and the domain SC and their shared phenomena a and b. The possible causes of hazardous behaviour in D are as follows:

- an anomaly in SC processing of a can cause a to cause hazardous behaviour in D
- an anomaly in D can cause b to cause an issue in SC which results in a causing a hazard in D
- an anomaly in SC processing of b can result in a causing a hazard in D.

These anomalies that cause hazardous behaviour in D involve the domains D, SC and their phenomena a, b. However, as noted above, the relationship between the domains and the phenomena is not explicitly covered by an FFA.

It appears that an improved safety analysis could be obtained if the relationship between the domains and their shared phenomena is analysed in detail. This suggested the use of a HAZOPS style of analysis based on applying the guide words to the phenomena. A proposed set of HAZOPS guide words is shown in Table 5.1.

The first seven Guide Words in Table 5.1 are the "classic" words used in the original Chemical Industries applications of the HAZOPS technique [6], the last four are added to deal explicitly with computer processing and real-time processing systems. In theory, the last four could be covered by the original words (e.g. *Early* could be covered by *Part Of*), but it assists and improves the analysis task to make them explicit. These four words are usually included in the analysis of computer-based systems [124].

Table 5.1: HAZOPS Guide Words

Word	**Description**
No/None	No part of the intention achieved
More	Quantitative increase in a physical property
Less	Quantitative decrease in a physical property
As Well As	All intentions achieved, but with additional effects
Part Of	Only some of the intention is achieved
Other Than	A result other than the intention is achieved
Reverse	The exact opposite of the intention is achieved
Early	Intention occurs before it should in time
Late	Intention occurs after it should in time
Before	Intention happens before it should in sequence
After	Intention happens after it should in sequence

Table 5.2: HAZOPS Applied to *int*

Word	**Local Effect**	**System Outcome**
No/None	*int* not active when it should be	No defensive release
No/None	*int* active when it should not be	Inadvertent release
As Well As	*fire* active & *int* fails active	Inadvertent release
Part Of	Intermittently active *int*	May delay or inhibit defensive release
Part Of	Intermittently inactive *int*	May result in inadvertent release
Other Than	*int* value indeterminate	DR: if *int* indeterminate then failsafe to inactive
Reverse	N/A	N/A
Early	*int* value active before it should	Inadvertent release
Early	*int* value inactive before it should	No defensive release
Late	*int* set active later than it should	Release delayed
Late	*int* set inactive later than it should	Possible inadvertent release
After	*int* after *fire*? and *sel*	No release when expected
Before	*int* before *fire*? and *sel*	Expected behaviour

As an example, consider the application of HAZOPS to the Decoy Controller model of Problem P_{Red} in Figure 5.2. Consider applying the guide words to the phenomena *int*. The results are shown in Table 5.2.

The HAZOPS analysis is more detailed and more specific than the FFA, and highlights the relationships between causes and the system effects they excite. For example, in Table 5.2 the second entry for *Late* is concerned with "*int* set inactive later than it should be". The system effect is a possible inadvertent release. The phenomena *int* should be set inactive when one of the interlocks *air*, *ok* or *out* becomes inactive. It is potentially hazardous to allow a release when any of these interlocks are not set to indicate it is safe to release. Therefore a delay in setting *int* from active to inactive could cause a hazard because a release could occur (since

int is active) even though one of the interlocks indicates that release should be inhibited. Therefore, an *int Late* anomaly can cause an inadvertent release, and the effect *inadvertent release* is directly associated with its cause *int set inactive later than it should be* by the analysis.

The advantage of the HAZOPS analysis direct cause to effect relationship is that it facilitates the identification of possible mitigations strategies. For example, in the case when *fire* is active and then *int* fails active (it should be inactive, inhibiting releases) then this would result in an inadvertent release. A typical mitigation is to require that a release can only occur if *int* was active prior to the *fire* request command, otherwise release is inhibited. Therefore with this mitigation *int* failing active after a *fire* would not cause a release. There would also need to be ongoing BIT (built in test) to identify the *int* failure and to then inhibit releases until it has been rectified.

The goal of the HAZOPS is to cover all possible anomalous behaviours of the system under analysis by careful application of the guide words. Implicit in the analysis approach is the assumption that if the HAZOPS is properly organised then the guide words **can cover all** the possible anomalous behaviours. This is supported by experience of using the technique on many applications [6, 106]. Further, the use of a diverse team of experts coordinated by an experienced moderator skilled in the use of HAZOPS maximises the chances that the **can cover all** is actually achieved.

A process for using FFA, HAZOPS and FTA in combination was developed as follows. The FFA was used to identify the hazardous behaviours of a domain (and cover internal phenomena) and HAZOPS applied to the phenomena shared between domains (based on the POSE model). Then FTA was used to investigate specific entities of interest identified by the FFA and HAZOPS work.

5.2.1.2 Improving a Formal PSA

The work on the *DC* case study has demonstrated that POSE can be used with the traditional safety analysis techniques such as FFA and FTA to achieve useful results. However, using POSE in combination with a formal technique opens up further opportunities for integrating the modelling with the safety analysis task to achieve a significant improvement in the safety analysis capabilities of the development.

Inductive techniques such as FFA are often used to identify possible hazards. In a formal analysis simulation would be used to achieve the same result. The goal of the formal development in this work is to use the POSE problem model to derive a formal behaviour equivalent using Alloy [52].

As noted above, completeness in the HAZOPS on a POSE model relates to the careful application of the guide words to the interface phenomena. In the formal model this translates into selecting representative values of the interface phenomena in time and space that will drive the behaviour of the system through its range of significant, possible behaviours. These behaviours are then checked to identify any problem cases which can be investigated using the proof tools. Fortunately, Alloy models are finite and the models developed from the POSE software problems are reasonably compact with small enumerations, which places an upper limit on the size of the analysis task. The role of the guide words in HAZOPS is to cover all possibilities, in the POSE/Alloy modelling the exhaustive nature of the simulation ensures the coverage/completeness goal is satisfied to a known degree.

The alternative approach is to use the Alloy proof capabilities to check specific safety issues. This is equivalent to a deductive analysis like FTA, and yields a proof that the Alloy specification model satisfies the specified proof condition. This is very strong, powerful, evidence to support the safety case, but does require that suitable proof conditions are identified *a priori*. Also, it is not always straight-forward to represent the required logic conditions adequately in the Alloy language. In practice,

the proof conditions are based on the high level safety analysis requirements.

In summary, formal analysis of the model can be tailored to achieve similar results to traditional safety analysis techniques, but with the added power of formal analysis and proof. However, as noted by Turski, there is always care needed to relate the formal results to the reality of the real world.

5.2.1.3 Safety Analysis Summary

This section has identified a process that allows FFA, HAZOPS and FTA to be used in combination to achieve an effective safety analysis using the POSE software problem model as its basis. The use of this process is evaluated in the *DC* case study safety analysis presented in section 5.2.2. This section also indicated how formal analysis could be used to achieve safety analysis based on proof and simulation. The use of formal techniques will be covered in the *SMS* case study covered in section 5.5.

5.2.2 Safety Analysis of the *DC*

A PSA using FFA and FTA was applied to the model P_{Exp} in section 4.4 and the results are recorded in the PS2 problem transformation step. The application of the PPT has removed domains and simplified the model to P_{Red} and this has been used to derive a specification for the *DC*. The aim of this section is to evaluate the usefulness of performing the PSA after the simplification on the model P_{Red}, and using the combined analysis process of FFA, HAZOPS and FTA described in section 5.2.1.1. Note that the same model used for the derivation of the specification P_{Red} is also used for the PSA – thus providing support for property **[SameA1]**.

FFA was applied to the function of each of the domains, and this include consideration of any internal behaviour functions and phenomena. Then HAZOPS was applied to the shared phenomena between domains and some of the interesting results for the *SC* domain are shown in Table 5.3.

Table 5.3: HAZOPS Summary for the *SC*

Id	Guide	Effect	Outcome
H1	No/None	No *fire*?	Release inhibited
H2	No/None	No *int*	Release inhibited
H3	No/None	No *fire*?	Release inhibited
H4	As Well As	Continuous *fire*?	**Inadvertent release**
H5	As Well As	Continuous *fire*	**Inadvertent release**
H6	Part Of	Intermittent *fire*?	Could inhibit release
H7	Part Of	Intermittent *fire*	Could inhibit release
H8	Part Of	Part of *int* not set	Release inhibited
H9	Early	*fire*? when not required	**Inadvertent release**
H10	Early	*fire* when not required	**Inadvertent release**
H11	Late	*fire*? when not required	**Inadvertent release**
H12	Late	*fire* when not required	**Inadvertent release**

The hazardous outcomes are shown in bold. The results were more comprehensive than those obtained using just FFA and FTA on P_{Exp}, and included all the hazards found from that analysis which were further investigated using FTA as before. The results demonstrated for this example that the combination of FFA, HAZOPS and FTA was effective.

A functional FTA applied to *DM* (based on the model of Figure 5.2) and using the HAZOPS problem cases (H4, H5, and H9 - H12) indicate that a failure in the uP (systematic or probabilistic) could result in the *fire*? failing on, which results in the same outcome as described in section 4.4. It is repeated here for completeness. The *Pilot*'s allow input provides some mitigation, but as soon as this is set ($ok = yes$) a flare will be released, which is undesirable behaviour. In other words, with this system architecture, H_2 is only protected by the *Pilot*'s allow input. If *fire*? failed on, then as soon as the *Pilot* indicated an intention to allow flare release, by selecting the switch, then the flare would be released, which is not the design intention. Therefore this safety analysis (as in the PSA reported in the PS2 of section 4.4) indicates that *fire*? needs to have a safety involved (not critical) integrity. **The current design does not satisfy the safety requirements.**

Another advantage shown by this work was applying the PSA to the reduced model P_{Red} and its simpler domain structure made the safety analysis task easier.

The original analysis had to trace through the effects of the domains that have now been removed, which complicated the analysis task. The tracing through in the original analysis was informal. In contrast, tracing through to determine an outcome is still required for the reduced model, but the assumptions associated with each problem progression make this task more systematic and easier to apply.

The conclusion after the PSA is that the current system architecture is not a suitable basis for the rest of the development.

5.3 Traceability and Backtracking: [TraceI4]

Traceability is a crucial feature of safety development practice (as discussed in section 3.2.3) and safety standards such as DS 00-56 [125] and the DO-178B [108] guidelines have stringent requirements with respect to demonstrating traceability from the requirements, through the design and onto the testing and analysis that validates the requirements.

A summary of a POSE development can be presented in graph form as [41, 85] where the nodes represent problems and the arcs are justifications of the groups of transformations that lead to the problems. In this work the transformations are grouped into problem transformation steps (henceforth steps) so each node is also a step in the development process – similar groupings are used in [41, 85]. This step information and the requirement associated with it can be annotated onto the graph to provide a richer representation of the development path where each node identifies:

- the software problem name
- the problem transformation step name
- the requirement that results from the step

and as before the arc is the justification that enables the step changes. For example, the graph for the *DC* problem development up to this point is shown in Figure 5.4. For instance, the tracing of the development of the requirement required by safety standards can be traced through the nodes and the strength of the argument to support the transformations evaluated by inspecting the justifications on the arcs. In Figure 5.4 the last step in the graph (PS6) produced P_{Red} and includes the PSA which demonstrated that the selected system architecture was not a suitable basis for the rest of the development – i.e. it is a dead-end in the development path.

The result of reaching a dead-end is the need to iterate back to evaluate alternative architectures, but guided by the knowledge of why the original architecture failed. A graph for the *DC* development is shown in Figure 5.6. The dead-end in the development records why a path has failed – useful information for any subsequent maintenance or upgrade phases to avoid repeating the mistake. In addition, the graph assists the task of finding out where to iterate back to. For example, in the *DC* graph of Figure 5.6 the reason for the dead-end at step PS6 was that the architecture failed the PSA and the iteration (as shown) is back to the point where the architecture was introduced, the start of PS2 in the right-hand path of the figure.

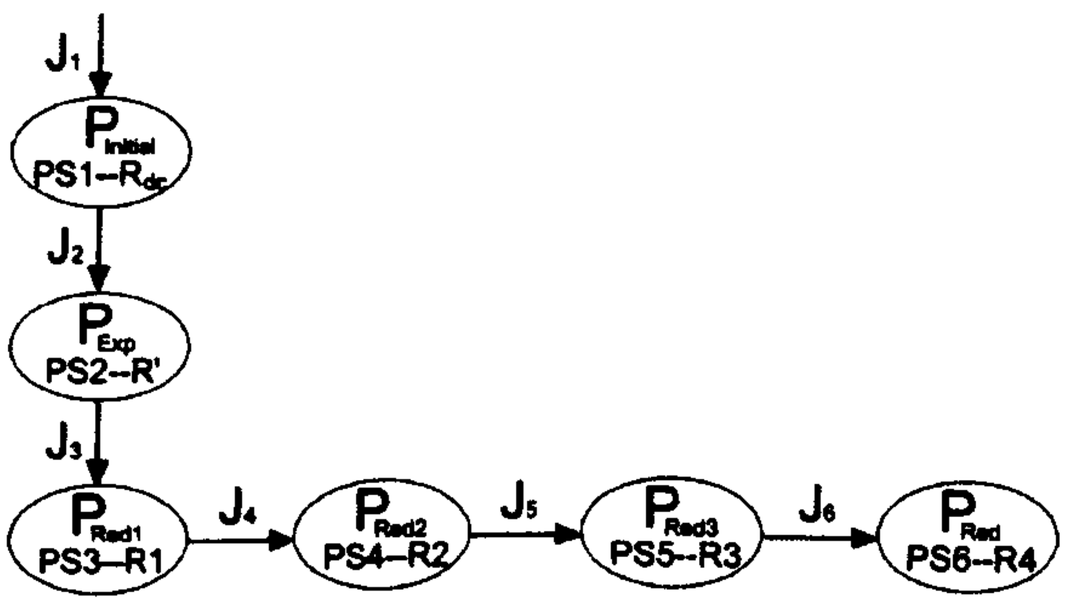

Figure 5.4: *DC* Problem Transformation Trace Graph

The design of the modified system architecture can be undertaken using the POSE AStruct. In the original development the *II* and *DM* were existing components and only the *SC* had to be designed (see section 4.3); in the modified

architecture both the SC and DM' need to be designed, so this is a *codesign* problem as outlined in [41]. The PSA identified that the *fire*? phenomena should have a safety involved integrity, and the preferred architecture solution was to separate out the *fire*? functionality from the non-safety involved timing and BIT functionality. The modified AStruct that supports this has the form:

$$DecoyContAS'[II^{int}_{ok,air,out}](DM'^{sel,fire?}_{con}, SC^{fire}_{int,fire?}).$$

In fact this is a simple codesign problem as the shared phenomena do not change, i.e., the redesign is internal to the DM' as shown in Figure H.2 in the problem transformation step PS2' shown below. A PSA was applied at step PS6' and the results were successful.

PS2':Backtracked application of SOLUTION INTERPRETATION AND EXPANSION *to* $P_{Initial}$

JUSTIFICATION J'_2: The newly identified system architecture, its components and relevant properties are summarised below (where they differ from J_2):

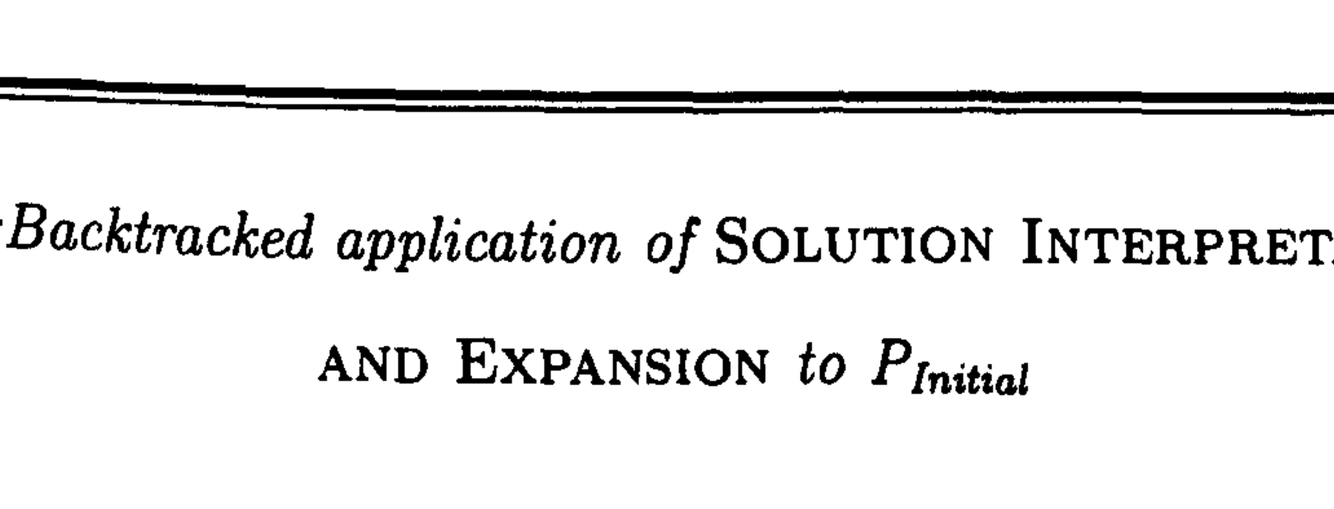

Figure 5.5: Modified DM' Architecture

The architecture consists of a message buffer (MB), a field programmable gate array ($FPGA$ [49]) and a microcontroller (uP) as shown in Figure H.2. The message buffer MB holds the received control message *con*; the micro-controller uP decodes it to extract the selected flare type (leading to *sel*); the FPGA component decodes it to extract a fire command request (leading to *fire*?).

INCLUDES: Includes J_2, with alterations as discussed below.

CLAIM: *The choice of candidate solution architecture exhibits sound safety engineering judgement*

ARGUMENT & EVIDENCE: The chosen system architecture is similar to the previous one (see J_2) except that as a result of the PSA we require the *fire*? signal to be safety involved (but not safety critical) so as to allow the overall architecture to satisfy its safety target. We do this by taking the safety involved functions out of the uP component and route them through a separate high integrity path. As a result, we choose a new component DM' in which there is a partition between the safety and non-safety elements: the simple safety functions (those associated with the *fire*? request) are routed separately through MB and $FPGA$, while the other complex functionality is routed through MB and uP. This means that only MB and $FPGA$, which have simple functionality, have to be designed to a safety related standard.

CLAIM: *The chosen solution architecture does not prevent the satisfaction of R.*

ARGUMENT & EVIDENCE: **The safety analysis (PSA) is applied to the revised system architecture and is successful - the is claim is supported.**

Another benefit is that the graph shows what steps need to be repeated. In this case five steps need to be repeated, but this process is assisted by the fact that much of the information from the earlier analysis can be re-used. The second iteration of the POSE process is similar to the first: although there is new information associated with the revised system architecture, the remainder of the transformations may be carried across from the first iteration with minimal change (just change of name), simplifying this second (and any subsequent) iteration. The second candidate architecture differs from the original in that we replace DM with higher integrity component DM'. The problem transformation development step is shown in PS2' and its associated annotated problem transformation graph is shown as the

right-hand path in Figure 5.6, where apart from P'_{Exp} the others are equivalent, e.g., P_{Red1} is the same as P'_{Red1} apart from a name change caused by introducing the prime character (i.e., DM to DM') and so on. The left-hand sequence in Figure 5.6 shows the original graph, backtracking to the PS2 step (on the right-hand side) after the unsuccessful PSA.

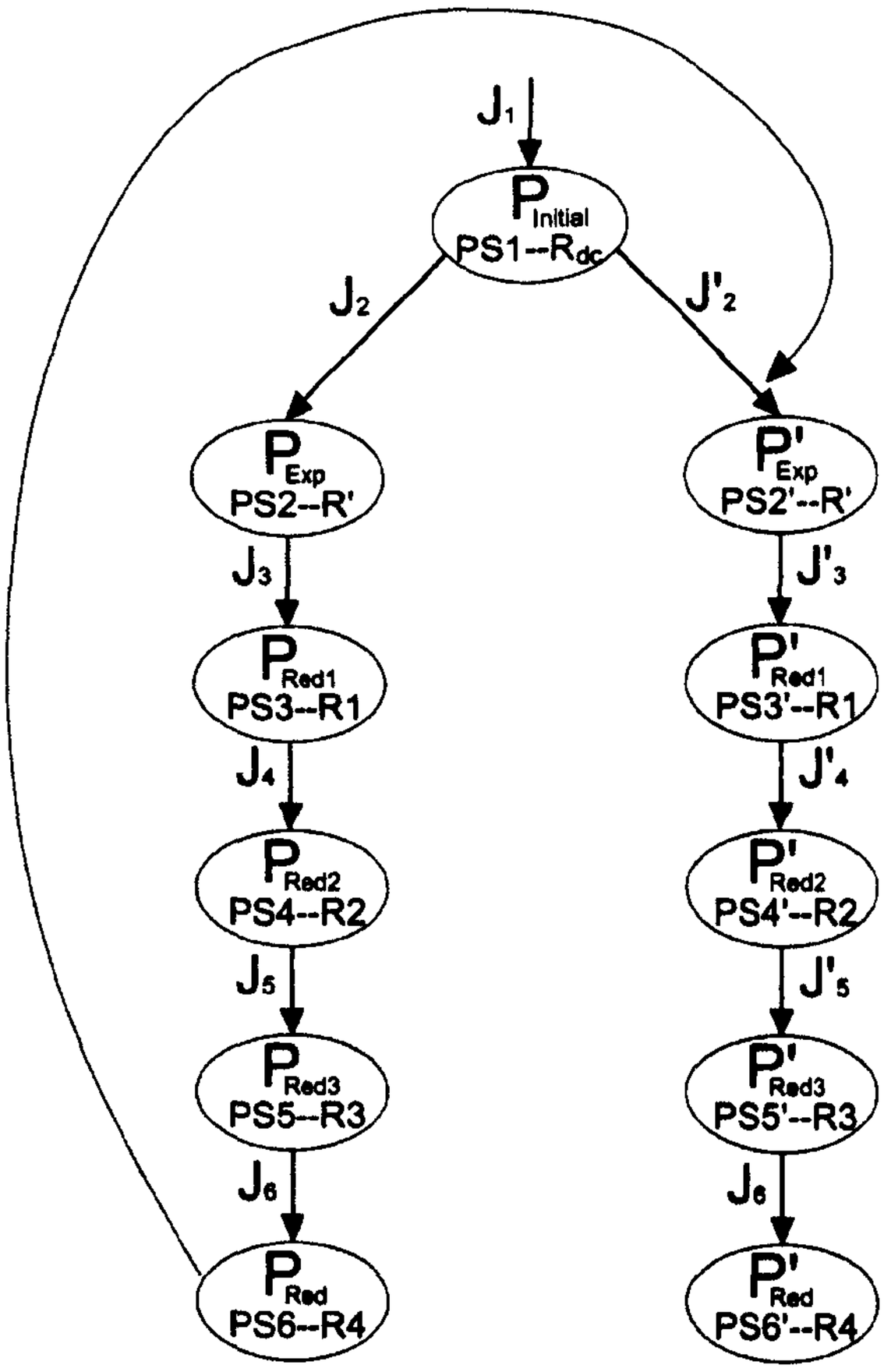

Figure 5.6: *DC* Revised Problem Transformation Trace Sequence

5.4 *DC* Case Study Discussion

The repeated application of the PPT using the process detailed in section 5.1.2 allowed the domain structure to be simplified, which facilitated the safety analysis task - supporting the properties of **Simplification** and **[SameA1]**, as the same model P_{Red} was used for the safety analysis as for the development. The fact that

similar results were obtained from the earlier more complex safety analysis and the safety analysis based on the reduced problem P_{Red} gives some strong evidence to support the validity of the problem progression process. In fact in the more complex analysis task the analyst had to work through the domain behaviour links in an informal manner. With the PPT, the domains are removed systematically and the necessary behaviour captured as assumptions. This was found to facilitate the analysis task by providing a structured trace between the domains, and was superior to the informal approach used earlier.

The *DC* high level functional requirement represented by *Ra* to *Rd*, are appropriate in that they refer to phenomena in the environment – i.e. they address the problem, not the solution. However, the goal is to design an *SC* that can satisfy these requirements. As shown in section 5.1.1, this requires the requirements to be made feasible, and section 5.1.2 demonstrated that the POSE PPT was capable of achieving this goal. P_{Red} is the simplified software problem after repeated application of the PPT to the *DC* system, and inspection of requirement *R4* in PS6 (see Appendix B.3) shows that the derived functional requirements (**R4a** to **R4d**) concerned with the *SC* are expressed in terms of phenomena directly accessible to the *SC*. That is, the derived requirements in *R4* are feasible and can be directly implemented. Inspection also shows that the specification can now be derived directly, supporting the properties **[NecessaryP]** and **Spec. Join**.

The combination of FFA, HAZOPS and FFA using the process outline in section 5.2.1.1 provides an effective analysis that produces comprehensive results when applied to a POSE software problem model, providing support for the effectiveness of the safety analysis aspects of issue **[LateI3]**.

The POSE problem transformation graph is a useful mechanism for identifying the steps involved in the development, tracing the requirements as they are transformed through the steps, and locating where to backtrack to when a dead-end is reached in a particular development path. As noted in the *DC* case study example,

only the step PS2 needed to be updated to form PS2', the other steps could be re-used (apart from the name change). This means that this POSE problem transformation graph provides the early phases of IPDP with a useful iteration facility, as well as providing evidence for mitigating **[TraceI4]**.

In summary, *DC* case study has provided support for the properties **[NecessaryP]**, **[SameA1]**, **Simplification** and **Spec. Join**; and the issues **[LateI3]** and **[TraceI4]**.

5.5 Combining POSE and Alloy [FormalA2]

This section investigates how POSE can be used with a formal method to provide a high integrity development process in support of property **[FormalA2]**, and how this combination can provide an improvement to an existing safety critical development process based on Z [116] and embedded into the IPDP described in Chapter 3. As introduced in Chapter 2 a requirement was that the formal modelling must support simulation and proof in support of the property **Validation**. Moreover it must be compatible with Z - given that the original development team were experienced in its use. This suggested the use of the B method [111] or Alloy [52].

Alloy was selected as the tool is freely available and well supported. Alloy is a lightweight formal method developed from the goal of combining the power of a SAT (Boolean satisfiability) solver with the descriptive power of the Z language. It has an active and growing user community and is continuously being developed and enhanced. It has strong simulation and proof capabilities within well-defined limits, and allows complex behaviour to be modelled using clear, simple constructs. As such, it fits well as the modelling tool for use with the POSE transforms.

The case study was concerned with examining the use of the POSE/Alloy combination in improving the early life cycle phase performance of a Z-based safety critical development process used in the author's Company (see section 3.4.2). The original

validation process correctly identified some subtle anomalies with the definition in Z of the high level safety properties, but these were found late in the process and led to substantial changes. The role of this case study is to address the safety questions:

(a) *could* these anomalies have been discovered earlier using POSE and Alloy, and

(b) *would* they have been discovered earlier by following a *reasonable* process based on POSE and Alloy.

In this context, a reasonable process is one in which nothing special or specific is done to identify the anomalies.

The main focus of this case study is the use of POSE in combination with Alloy to address the property **[FormalA2]**. Therefore the development of the problem steps will be summarised in the main text, with the detail being presented in Appendix D.

5.5.1 The Case Study

The case study investigated in this section is based on a Stores Management System (SMS) and focuses on its selective jettison (SJ) functionality, i.e., the way in which stores are chosen for jettison from the aircraft. Figure 5.7 shows a schematic view of the six release/jettison stations on the aircraft wings together with the *SJ* panel which is used by the pilot to make jettison selection choices. The *Pilot* can select *SJ* to be off, or jettison of all stations (Ship mode) or jettison of selected stations (Station mode).

The *SJ* sequence does not release all the stores at once, as this could result in a collision hazard, rather the step release sequence used is `Out` → `Mid` → `In` with a delay between each step. That is, the *SJ* is not an atomic action, but rather a sequence of three atomic actions. A balance algorithm is applied to most jettison sequences to ensure that the aircraft is not put into an unsafe state. The exception is that a single store jettison is always allowed to give the pilot final control under error conditions.

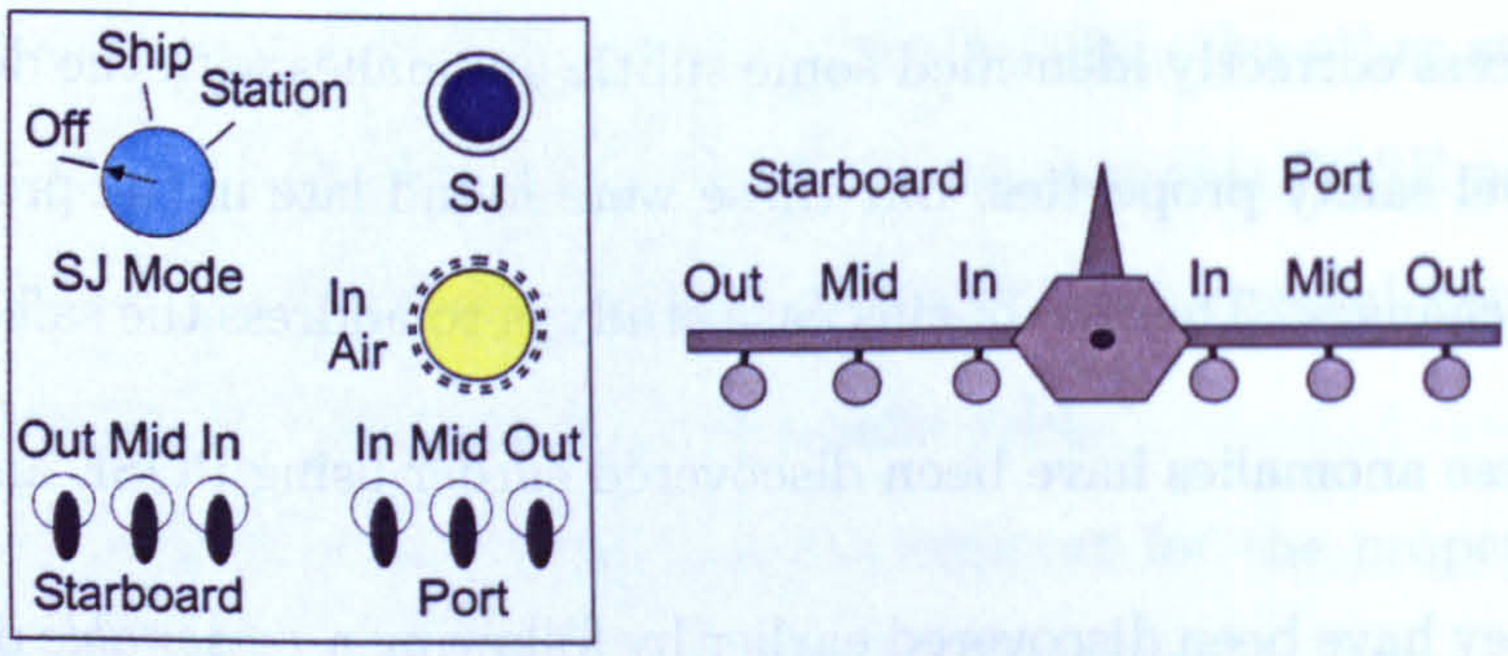

Figure 5.7: *SJ* Panel (*SP*) and Aircraft Schematic

The above information and more is collated as part of the application of the **SPS1:Initial Problem** problem transformation step using a similar approach to that described in section 4.2 for the *DC* case study. This captures understanding the environment context, the requirements and the initial system architecture in agreement with the early phases of IPDP. Note that as with the *DC* case study, the requirements are based on those that were originally presented and have not been put into a good practice form (the case study in Appendix H shows requirements in an appropriate good practice form). The resulting SPS1 problem transformation step is presented in Appendix D and leads to the initial software problem of:

$$P_{Initial}: \begin{array}{l} S\&RE(jet\_stat)_{jetv},\ Pilot(p\_mode, p\_sel, p\_jet)^{smode,sstat,ssj}, \\ SMS(InAir)^{jetv}_{smode,sstat,ssj} \end{array} \vdash R^{jet\_stat}_{p\_mode,p\_sel,p\_jet,InAir}$$

In this case study care was taken to ensure that the requirements were at an appropriate level of abstraction using the approach outlined in section 5.1.3 whereby a check was applied to ensure that each requirement referred to phenomena that were adjacent to it.

The next step involves exploring possible solution system architectures and follows a similar format to that used on the *DC* case study in section 4.3. The result

is the problem transformation step SPS2 reported in Appendix D, which leads to the software problem:

$$P_{Exp}: \quad \begin{array}{l} S\&RE(jet\_stat)_{jetv},\ Pilot(p\_mode, p\_sel, p\_jet)^{smode,sstat,ssj}, \\ SP(InAir)^{invec}_{smode,sstat,ssj},\ SU^{jetv}_{outv},\ SM^{outv}_{invec} \quad \vdash \quad R'^{jet\_stat}_{p\_mode,p\_sel,p\_jet,InAir} \end{array}$$

The step from $P_{Initial}$ to P_{Exp} also transforms the requirement from R to R' as shown in the model. The PF diagram form of this model is shown in Figure 5.8.

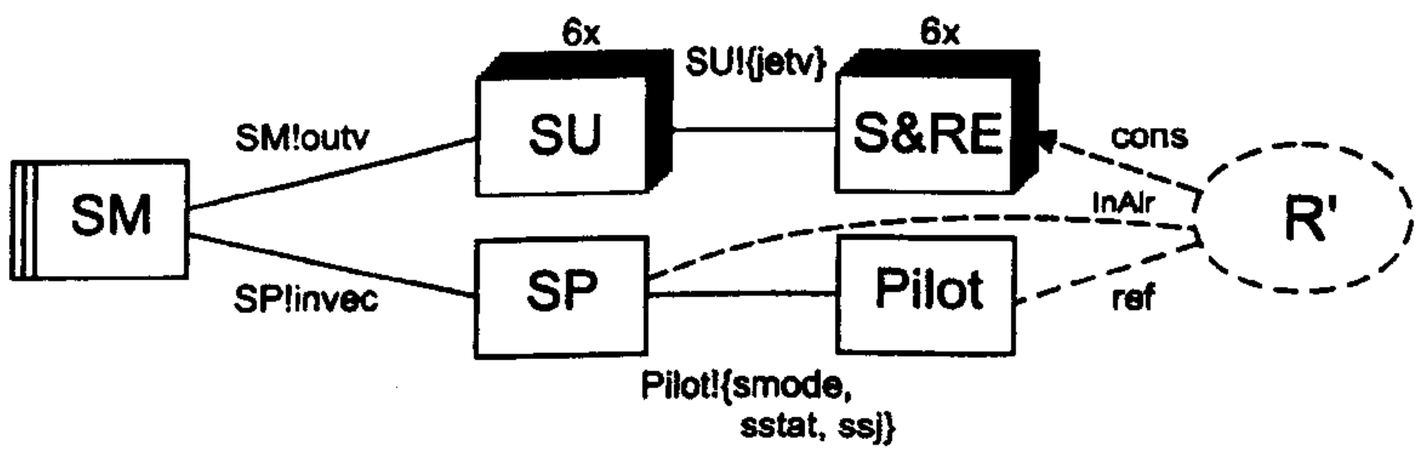

Figure 5.8: *SMS* Expanded Problem, P_{Exp}

5.5.2 Simplification Through Problem Progression

Inspection of the software problem P_{Exp} shows that there is a gap between the requirements R' and the machine to be designed, *SM* (see also Figure 5.8). The requirements phenomena are $cons = (jet\_stat)$, $InAir$, $ref = (p\_mode, p\_sel, p\_jet)$, whilst the *SM* phenomena are *invec*, *outv*. This indicates the need to apply the PPT to simplify the problem to support the PSA and transform the requirements to be in terms of the *SM* phenomena so that the specification of the *SM* can be derived. The first domain to be removed is the *Pilot* – it fits the criteria of being adjacent to the requirements. It is shown in detail as SPS3 in Appendix D. At this point, after removing the *Pilot* domain, the software problem has been transformed into:

$$P_{Red1}: \quad S\&RE(jet\_stat)_{jetv}, SP(smode, sstat, ssj, InAir)^{invec}, \quad SU^{jetv}_{outv}, \quad SM^{outv}_{invec} \vdash R1^{jet\_stat}_{smode,sstat,ssj,InAir}$$

Note that the requirements have been transformed from R' into $R1$, all the SPS3 claims are successfully discharged, and that as in the earlier problem transformation steps, these claims and their discharge evidence correspond to important elements of the safety case argument.

So far the domain removals have all been reference-type PPTs since they cover input data into the system. The six $S\&RE$ domains are different as they represent the output of the system (the actual release of stores) and are therefore constrain-type PPTs. Although there are six $S\&RE$ domains, they are identical in functionality and can be treated as a single parametrisable domain in terms of applying the PPT to effect domain removal. The problem transformation step SPS4 follows a similar format to SPS3 and covers the $S\&RE$ domain removals. The details of these PPT applications are presented in Appendix D, where the requirements transformation from $R1$ to $R2$ is presented in some detail to demonstrate the steps involved. This simplification work provides further support for property **Simplification**.

The software problem model after removing the $S\&RE$ domains is given by:

$$P_{Red}: \quad SP(smode, sstat, ssj, InAir)^{invec}, SU(jetv)_{outv}, SM^{outv}_{invec} \vdash R2^{jetv}_{smode,sstat,ssj,InAir}$$

and this corresponds to the requirement R2. The PF diagram form of this problem is shown in Figure 5.9.

The sequence of problem transformation steps to proceed from the initial problem $P_{Initial}$ through to the reduced problem P_{Red} can be conveniently represented in the

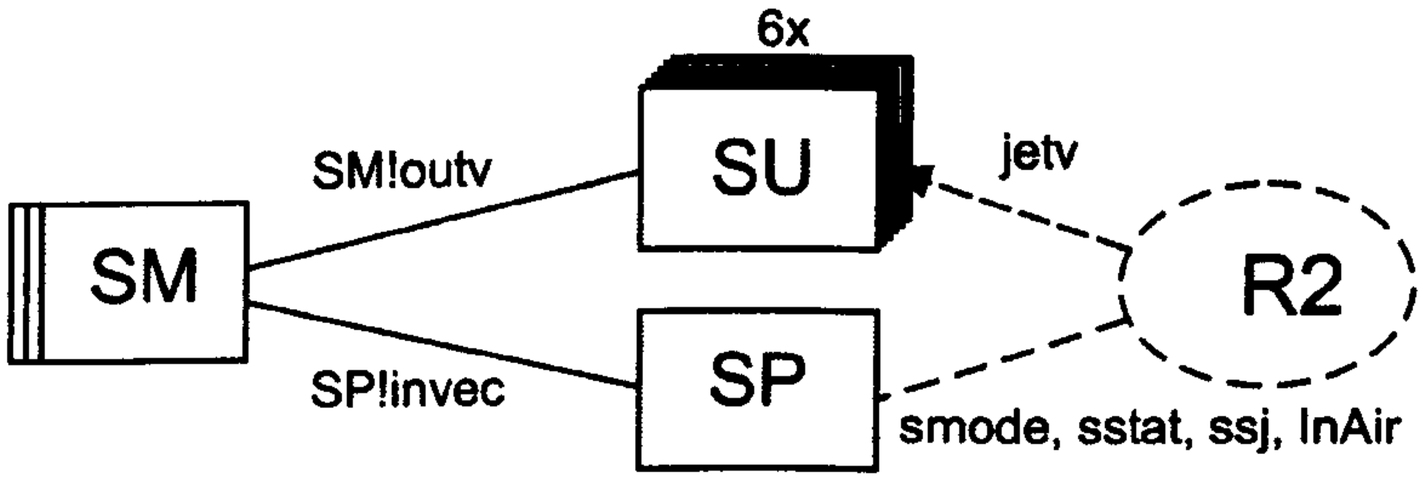

Figure 5.9: *SMS* Problem After *S&RE* Domain Removals

problem transformation graph shown in Figure 5.10.

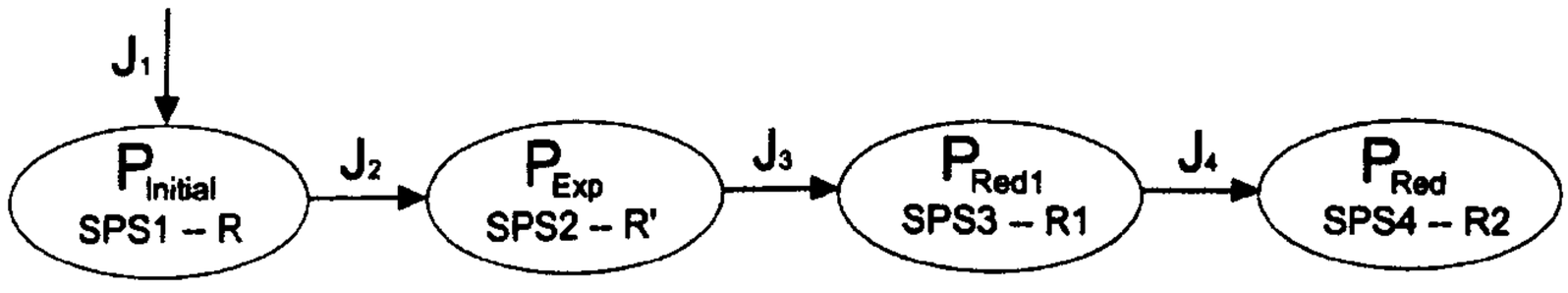

Figure 5.10: The *SMS* Problem Transformation Graph

5.5.3 Deriving and Validating the Machine Specification

The POSE model represented by problem P_{Red} and its equivalent PF problem diagram form of Figure 5.9 are almost at an appropriate level of abstraction to support the safety analysis and modelling work. Inspection of these representations indicates there is still a small gap to be resolved between the requirements expressed in terms of the phenomena *jetv*, *smode*, *sstat* and *ssj*, and the machine to be developed, *SM*, that interfaces to the phenomena *outv* and *invec*. There is a need to relate *outv* to *jetv*, and *invec* to *smode*, *sstat*, *ssj*, and *InAir*.

We could apply the PPT again to remove the *SU* and *SP* domains, since this would relate the resulting requirement phenomena directly to the machine *SM* phenomena. However, the resulting model is not as useful as P_{Red} for the PSA work. A useful way forward is to apply REQUIREMENTS INTERPRETATION to the *SU* and *SP* domains to map the phenomena adjacent to the requirement *R2* to the phenomena adjacent to the machine *SM*. If the relationship involves only Environment Constraint behaviour (i.e., the only difference between the phenomena can

be directly resolved by reference to the behaviour of the intervening domain the phenomena map through) then this mapping can be used to derive the machine specification and to validate that it satisfies $R2$. In effect, REQUIREMENTS INTERPRETATION is applied to obtain the useful behaviours of the intervening domains which is used to establish the relationship between the machine and the requirement phenomena – this mapping works if the behaviour of the domains corresponds to the Environment Constraint non-feasible requirement. The mapping confirms the correspondence between the phenomena as *outv* is equivalent to *jetv* and *invec* to $(smode, sstat, ssj, InAir)$.

In effect, *jetv* and *outv* are functionally equivalent representations. With these relationships established, it is now feasible to derive the specification of the SC directly from the POSE entailment of problem P_{Equiv}, which is P_{Red} with *outv* substituted for *jetv* and *invec* substituted for $InAir$, $smode$, $sstat$ and ssj in direct support of properties [**NecessaryP**] and **Spec. Join**. The resulting problem is:

$$P_{Equiv}: \quad SP(invec)^{invec},\ SU(outv)_{outv},\ SM^{outv}_{invec} \vdash R3^{outv}_{invec}$$

and the corresponding requirement, $R3$ is:

R3a The SM shall ensure the safe selection of stores through the SU **phenomena** *outv* when commanded by the appropriate settings **in** *invec*.

R3b The SM shall have a Selective Jettison (SJ) facility which will **select all stores for jettison in** *outv* (Ship mode) or **select specific stores for jettison in** *outv* (Station mode) from the SU in accordance with *invec*.

R3c The SP shall allow each of the **six station selection positions in** *outv* **to**

be selected for *SJ* in accordance with the *invec*. In Station *SJ* mode, a station must be selected before it can be jettisoned.

R3d The *SP* shall have an In Air lamp **indication in** *invec*, which will be lit if the aircraft is in the air. *SJ* is only allowed if the aircraft is in the air.

R3e The *SP* shall have a *SJ* button **that allows jettison to be initiated in accordance with** *invec* and it must remain pressed for the duration of the jettison sequence. The following must hold prior to initiating the *SJ*:

1. *SJ* mode selected (Ship or Station) **in accordance with** *invec*.
2. Aircraft is in the air.
3. If Station mode, the appropriate stations are selected on the *SP* **in accordance with** *invec*.

R3f The *SM* shall control the selection of stores for jettison **in the** *outv* from the *SU*, by controlling the *SU*s to **correctly** provide conditioned power and drive signals **through its output phenomena** *outv*.

R3g The *SM* shall implement a Balance Algorithm which can remove items from an *SJ* package to ensure aircraft roll moments are contained within safe limits.

R3h It shall be possible to *SJ* a store from a single station without regard to the aircraft balance algorithm calculation.

R3i The *SJ* release sequence will be Out stores first, then Mid and then In.

At this point the phenomena of the machine *SM* are the same as the phenomena referenced by R3, which therefore represents the specification of the machine. The contribution from the *SP* domain is to confirm that *invec* directly relates to *smode*, *sstat*, *ssj* and *InAir*. The contribution from the *SU* domain is to confirm that *outv* directly relates to *jetv*. From this we have that:

Table 5.4: Definitions of *outv* and *invec* in Alloy

outv :	(i)	`relp`	Indicates a valid SJ sequence is in progress;
	(ii)	`pulses`	Defines which stations receive SJ pulses for this time instance;
	(iii)	`balan`	Indicates whether the SJ is to satisfy the balance condition.
invec :	(i)	`mode`	Indicates the SJ mode selection (off, Ship or station);
	(ii)	`sels`	Set defining the status of the SJ station selection switches;
	(iii)	`sj`	Indicates whether the SJ Button has been pressed;
	(iv)	`fl`	Indicates if the aircraft is in the air (air) or on the ground (grd).

$$spec.(R3) \wedge domaincontrib.(SU) \wedge domaincontrib.(SP) \Rightarrow requirement(R2).$$

Therefore the validation step is to demonstrate that the specification (R3) in combination with the domain contributions from SU and SP leads to the requirement ($R2$). Inspection of $R2$, $R3$ and the domain contributions confirmed the validity step informally. The next task was to develop a formal model and use formal analysis to confirm this validity step.

5.5.4 Development of the POSE/Alloy Model

The Alloy formal model was derived directly from the POSE problem P_{Equiv} and the first task was to define the *outv* and *invec* phenomena as Alloy variables. These are defined as in Table 5.4.

The phenomena `balan` (indicating whether the balance algorithm has been applied to the SJ package) is an implicit modelling parameter that is not an input to the SU. It is included in outv because it facilitates the model validation task. It is not required in the implementation because its effect is manifested in the `pulses` value – in fact its value can be derived from analysing the `pulses` value.

As noted earlier, the SJ sequence does not release all the stores at once, as this could result in a collision hazard, rather it releases stores in the step sequence `Out` → `Mid` → `In` with a delay between each step. This is captured by the requirement **R3i** and the use of `cnt` in the Alloy model. That is, the SJ is not an atomic action, but rather a sequence of three atomic actions in sequence.

If a set of heavy stores were jettisoned from only one wing, the aircraft would

be subjected to a potentially lethal roll moment. To avoid this problem, jettison packages (including Ship) are subjected to a balance algorithm calculation, which can remove items from the SJ package to ensure that the aircraft remains within safe roll moment limits. This is captured as requirement **R3g**. Store on Station (SoS) indications are used to signify if a store is present or not, these are used by the balance algorithm to make its calculations. However, it is recognised that these SoS indications could fail in such a way as to fool the balance algorithm. In particular, this might result in the balance algorithm inhibiting a jettison of a store when in fact jettisoning the store could result in a more benign residual roll moment. To counter this problem, the Pilot is allowed to SJ a single store, irrespective of the Balance algorithm. This is captured as requirement **R3h**.

The requirements specification Alloy model of the SM was developed directly from the POSE model, using the same interface and phenomena as used in P_{Equiv}. It is shown in full in Appendix E. The functionality of this Alloy model was based on the information collected in support of the POSE problem transformation step development from SPS1 through to SPS4 (it includes the domain description information and the existing Z specifications). In Alloy terms [52], the SM module makes use of an extension to the standard Alloy Natural numbers utility file. The extension just involves defining terms for `Two` to `Ten`, in addition to the `Zero` and `One` already provided. The full Alloy specification is outlined in [52], so the following presents the main points of the model relevant to our case study.

The signature (set) type definitions are based on an abstract set, with distinct single subsets. A typical definition for the type **SJMode** is shown below:

```
abstract sig SJMode {}
one sig  off, ship, stat extends SJMode {}
```

where **off** and **ship** correspond directly to there respective SJ mode switch settings and **stat** corresponds to station mode. Similar definitions were defined for (a) **SSet**

{son, soff} to represent the SJ button and whether a valid jettison release pulse is allowed, (b) StatsSel {s0, s1, s2, s3, s4} where each sn covers a particular SJ switch station selection set, (c) InAir {air, grd} to represent the In Air indication, (d) SPul {p0, p10,p01, p2} to represent the jettison release pulse patterns, and (e) BAL {nob, no_bal, is_bal} to represent if the SJ package is required to satisfy the balance algorithm. Where nob represents not applicable, no_bal represents balance algorithm not applied and is_bal means the balance algorithm is applied to the SJ package. These types are used to define the overall state of the SM as follows:

```
sig SMState { ntime  : Natural,
mode   : SJMode,     relp   : SSet ,
sels   : StatSel,pulses : SPul,
sj     : SSet,balan  : BAL,
fl     : InAir,      cnt    : Natural } {}
```

In SMState ntime represents the system simulation time reference and is modelled as a natural number, and cnt is a local time counter used for timing the SJ release sequence. Effectively, the other state components on the LHS are inputs from the SP and were defined using the SP! invec phenomena from the POSE model. The other RHS state components are outputs to the SU defined using the SM! outv phenomena. The relp = son means it is valid to provide SJ pulses, and pulses defines the pattern of pulses to be output at this time (note p0 means no pulses output). The initial conditions are defined by the predicate init[], which places the system in a known safe state. This is an important aspect of any system development and it also satisfies the initialisation concern identified by Jackson [53].

```
pred init (sm: SMState) {sm.mode=off && sm.sels=s0 && sm.sj=soff && sm.fl=grd &&
sm.ntime=Zero && sm.relp=soff && sm.pulses=p0 && sm.balan=nob && sm.cnt=Zero}
```

The trace model used is based on that used in Chapter 2 of [52] and involves using a linear ordering to order the traces with init as the first trace. In the following the

InVec() represents the input vector to the SM and Stim() represents the simulation time increment predicate. The trace timing simulation covers five slots from Zero through to Four. This can be extended as required.

```
fact traces {
 init[sms/first[]]
 all sm: SMState-sms/last[] | let sm'=sms/next[sm] | some m: SJMode, sel: StatSel, s: SSet, f: InAir |
 (InVec[sm.ntime,Zero,m,stat, sel,s0,s,soff,f,grd] && Stim[sm,sm']&& SMfun[sm,sm',m,sel,s,f]) or // 1
 (InVec[sm.ntime,One, m,stat, sel,s0,s,soff,f,air] && Stim[sm,sm']&& SMfun[sm,sm',m,sel,s,f]) or // 2
 (InVec[sm.ntime,Two, m,stat, sel,s1,s,soff,f,air] && Stim[sm,sm']&& SMfun[sm,sm',m,sel,s,f]) or // 3
 (InVec[sm.ntime,Three,m,stat,sel,s1,s,soff,f,air] && Stim[sm,sm']&& SMfun[sm,sm',m,sel,s,f]) or // 4
 (nat/gte[sm.ntime,Four] && m = off && sel = s0 && s = soff && f = air  && SMfun[sm,sm',m,sel,s,f])  }
```

The behaviour of the SM is defined by the predicate SMfun. This sets the valid SJ pulse indicator (sm'.relp=son), checks balance (balance(sm')) and outputs SJ pulses in sequence (pulTab(sm')) if a valid SJ button press is detected or the SJ sequence is still valid and continues. Otherwise, it resets the SM state to a safe value.

```
pred SMfun [sm, sm' : SMState, m : SJMode, sel : StatSel, s : SSet, f : InAir] {
(sm'.mode = m  && sm'.sj = s && sm'.fl = f && sm'.sels = sel)  &&

( ( ( sm.mode  = stat &&  sm.sels != s0 && sm.sj = soff && sm.fl = air &&
     m = stat && sel = sm.sels && s = son && f = air)
=> sm'.relp = son  && sm'.cnt = One && pulTab[sm'] && balance[sm']) &&

( ( sm.mode = stat && sm.sels != s0 &&  sm.sj = son && sm.fl = air && sm.relp = son &&
    m = stat && sel = sm.sels && s = son && f = air)
=> sm'.relp = son && sm'.cnt = incnt[sm.cnt] && pulTab[sm']  && balance[sm'])  &&

  ( not(m = stat && sel != s0 && sel = sm.sels && s = son && f = air)
=> sm'.relp = soff && sm'.cnt = Zero && sm'.pulses = p0  &&  sm'.balan = nob )
)  }
```

The function incnt() increments cnt, except if cnt is Three when it resets it to One. The predicate pulTab() is used to define the SJ pulse sequence timing, with sm.cnt=One corresponding to SJ from the Out stations, Two from Mid and so on. By varying the content of the s1 to s4 definitions, the complete range of SJ package

selections can be covered – the full definition of `PulTab()` is given in the Alloy code in Appendix E.

```
pred pulTab [sm : SMState] {

/* s1 -- Port Out & Starboard In selected for SJ */

(sm.sels = s1 && sm.cnt = One => sm.pulses = p01) &&

(sm.sels = s1 && sm.cnt = Two => sm.pulses = p0) &&

(sm.sels = s1 && sm.cnt = Three => sm.pulses = p10) &&


/* similar definitions for S2, S3 and S4 */


 /* s0 -- No stores selected for SJ */

(sm.sels = s0 => sm.pulses = p0)  }
```

The balance algorithm is covered by the predicate `balance()`, where if there are one or zero SJ pulses, then no balance (no_bal) otherwise balance is required.

```
pred balance[sm : SMState] { sm.relp = son &&

((sm.pulses = p10 or sm.pulses = p01  or sm.pulses = p0) => sm.balan = no_bal

    else sm.balan = is_bal ) }
```

5.5.5 Model Validation and PSA

Having produced the formal Alloy model of the specification ($R3$) – the full model shown in Appendix E – the next step is to validate that this model does satisfy $R3$. That is, we need to validate that the Alloy model adequately represents the specification $R3$. This was achieved by performing a variety of simulation runs, where each run used a different combination of `InVec()` values to explore the behaviour of the model over its range of inputs. The results validated that the Alloy model satisfied the specification given by $R3$.

The next task was to validate that the validated specification $R3$ (represented by the Alloy model) in combination with the useful behaviours (domain contributions)

from the *SU* and *SP* satisfied the requirement represented by *R*2. This being part of the process of validating that the machine specification *R*3 satisfies the original requirement *R* (as described in section 5.1.2).

The useful behaviour relations (domain contributions) of the domains *SU* and *SP* are introduced into the Alloy model as Alloy facts (using Alloy "fact" construct and model structure respectively) since they represent indicative properties of the environment context. In contrast, the optative behaviour of the requirement, *R*2 is introduced using predicates, and these predicates were proved using the Alloy "Assert" mechanism. The end result was validation that the specification (*R*3) satisfies its requirement (*R*2).

After validating that the Alloy model adequately represented the specification, and then validating that this specification satisfied its requirement, there is now enough supporting validation evidence to make it reasonable to apply the PSA. The PSA task has two main components to cover the complex functionality aspects represented by the Alloy model, and the more straightforward behaviour of the existing hardware components. The latter are addressed using the non-formal approach outlined in section 5.2.1.1, and is not considered further in this case study work. The complex behaviour is addressed using the approach outlined in section 5.2.1.2 and this work concentrates on these issues. Simulation was used to investigate the behaviour of the model looking for cases that violated one of the safety requirements. For example, consider the original requirement for *Rg* (see Appendix D):

Rg The *SMS* shall implement a Balance Algorithm which can remove items from an *SJ* package to ensure aircraft roll moments are contained within safe limits.

qualified by *Rh* which always allows a single SJ. From these can be derived a hazard "*Multiple station SJ (Rh not active) but balance not applied (Rg issue)*".

Simulation of the Alloy model using the `s1` station selection set (refer to InVec() definition above) indicated that `balan = no_bal` for this set. However, `s1`

is a multiple SJ—Port Out and Starboard In, so the release pulse sequence should be p01;p0;p10 for cnt of One, Two and Three respectively—so balance should be applied. Simulating with other station selection sets indicated that balance was only applied when more than one pulse was required at a particular time (i.e. p2) and only for that time. The required behaviour is that balance should always be applied unless the SJ package consists of a single release (a single p10/p01 with two p0s). The problem occurs in the high level Z specification of the SM, which contains the term:

$$(\#releaseLocations > 1 = \theta \mathsf{SM} \in \mathsf{balanced})$$

where # means the cardinality of the set and θSM means the bound values of SM. Now releaseLocations is the number of jettison pulses being applied—this is modelled in Alloy by the function balance(), which specifies no balance only if there are zero (p0) or a single (p10/p01) pulse. The problem is that (as noted above) SJ is not an atomic action, but rather three: one each for Out, Mid and In. The functionality given above ensures each atomic action satisfies the balance, but does not ensure the entire SJ sequence does. Hence selection set s1 which results in the pulse sequence p01;p0;p10 meets the criteria for each of its atomic components, indicating that the model does not require it to be balanced, when it should be. The solution is to base the balance calculation on the SJ station selection and not the jettison pulses. Implementing this change, results in balance() being updated as follows:

```
pred balance[sm : SMState] { sm.relp = son &&
        (sm.sels = s4 or sm.sels = s5) => sm.balan = no_bal
         else sm.balan = is_bal  }
```

Simulating with this form resulted in the correct behaviour for s1 and the other selection set combinations. Aspects of this behaviour were then proved.

Other safety issues were addressed in a similar fashion, and the modelling iden-

tified the further known anomaly and a not previously identified one. The latter concerned an inconsistency between two of the Customer supplied requirements documents. In all these cases the implementation was checked and it was confirmed that it operated as intended. That is, the problem was with the requirements specification.

An alternative approach would have been to represent the hazard as a logical assertion and use the model to prove whether the model does or does not satisfy the condition. This is the second part of the formal safety approach described in section 5.2.1.2.

5.6 Discussion

The early work in the chapter has shown that the POSE PPT can be usefully applied to transform requirements to make them feasible and allow the specification S to be derived in direct support of property **[NecessaryP]** and also support for **Spec. Join**. This was supported with evidence from the case study work, where the *DC* case study (section 5.1.3) demonstrated the derivation and validation of the specification using non-formal techniques, whilst the *SMS* case study (section 5.5.3 onwards) did the same for a formal model, this validation providing support for the property **Validation** and mitigation of issue **[ValidI6]**. The formal model provided stronger validation evidence, but there is always an extra step corresponding to the need to relate the formal model to reality. In the *SMS* case study this involved the need to confirm that the Alloy model did represent the specification represented by *R3*.

In Chapter 4 the safety analysis was conducted on a more complex model involving all the domains (e.g. compare P_{Exp} and Figure 4.2 with P_{Red} and Figure 5.2) and relating the events at the machine interface to corresponding events at the system boundary involved establishing informal relationships. The advantages of the

PPT-based approach of this chapter are that the analysis is easier because a simpler model is used, and the effect at the system boundary can be established by tracing through the relevant domain assumptions, which are established and validated as part of the PPT application.

Application of the PPT transforms complex problem descriptions into simpler ones, therefore the PPT application provides support for the property **Simplification**. It provides other benefits, for example, the discussion about the *DC* case study in section 5.4 noted that an improved safety analysis and feasible requirements can be achieved through application of the PPT. The information gathered about what a domain contributes to the overall system function so that it can be successfully removed is just the information necessary to complete the validation of a successful analysis. The PPT gathers this information in a systematic way that provides the traceability required by the safety standards and guidelines and thus provides some mitigation for issue **[TraceI4]**. The traceability aspects are further enhanced by the use of annotated problem transformation graphs as described in section 5.3. A problem transformation graph was used in section 5.5.2 to depict the progress of the application of the problem transformation steps in the *SMS* case study.

Processes for improving the PSA using non-formal and formal techniques were introduced in section 5.2. These were used to support the PSA in both the *DC* and the *SMS* (section 5.5.5) case studies – providing mitigation for the effective safety analysis aspect of issue **[LateI3]**.

In both the *DC* and the *SMS* case studies the same model used for the development, the POSE software problem model, is also used for the safety analysis which provides evidence to support property **[SameA1]**. The case study work also introduced a check to ensure that the requirement was at an appropriate level of abstraction, providing support for the property **Right Abstraction**. This was used successfully in the *SMS* case study.

The *SMS* case study work agrees with the earlier case study that POSE is flexible in use, and integrates well with the overall development process – IPDP in this case. This ability to work with existing development processes is important, because it supports the evolution and improvement, rather than the replacement, of these processes. This is in line with the goal of improving the 'normal' design, and avoids the damaging 'radical' design cycle [131]. It also addresses the property **[IntegA3]** identified in section 3.4.5, and property **Integrates** from Table 2.3, of improving and integrating well with IPDP.

The task of using the POSE/Alloy combination to improve the front-end of a safety critical development process asked two questions. The first, and easiest to address, was whether the POSE/Alloy combination could detect the anomalies. The *SMS* case study results demonstrate that both POSE and Alloy can be used in combination to detect the anomalies of interest, and thus provides direct support for property **[IntegA3]**. The second question is whether the POSE/Alloy combination would have detected these anomalies. This is more difficult to answer because it is a process question. A plausible answer can be gleaned from the development of the POSE model from SPS1 through to SPS4, and then the use of this model to produce the Alloy model described in section 5.5.4. This work shows that POSE and Alloy were used as recommended and found the anomalies of interest. This imparts a certain degree of confidence that the process used with the POSE/Alloy combination would be able to detect these and similar anomalies, and at an early phase in the development life cycle. Therefore technically the POSE/Alloy combination could and would be expected to provide the required process improvement for the safety critical development process.

The development of the formal model in section 5.5.4, and the results obtained in section 5.5.5 of the *SMS* case study also demonstrate strong support for property **[FormalA2]**.

In summary, completion of the *DC* case study and the results of the *SMS* case

study both provided strong evidence to support the properties [**NecessaryP**], **Spec. Join**, **Validation**, [**ValidI6**], **Simplification**, [**TraceI4**], [**SameA1**], **Right Abstraction**, [**IntegA3**], **Integrates** and [**FormalA2**], and some support for issue [**LateI3**]. Also, the early problem transformation steps in the *SMS* case study provided further support for the issues [**EnvirI1**] and [**RequI2**], and for properties **Context**, **Architecture**, **Avoid Bias** and **Model Reality** as discussed in Chapter 4 for the *DC* case study.

5.7 Chapter Summary

The chapter began with an introduction to the POSE PPT (Problem Progression Transformation), developed a process for applying it and then showed how it could be used to derive a specification S by transforming high level requirements into feasible requirements that apply directly to the machine to be developed. The completion of the *DC* case study and the results of the *SMS* case study provided examples of the successful use of the PPT to derive S and also to facilitate the safety analysis task which used the same POSE model – providing evidence to support the properties [**NecessaryP**] and [**SameA1**]. The *SMS* case study also showed that POSE could support a formal development process as required by the safety critical standards and also in support of property [**FormalA2**].

The traceability and backtracking (iteration) capabilities of POSE were developed in section 5.3 and shown to be effective at managing the backtracking required in the *DC* case study, providing evidence to support [**TraceI4**]. The case studies also demonstrated the usefulness of the safety process improvements investigated in section 5.2.

In summary, this chapter has developed some of the POSE capabilities such that there is now evidence that this expanded POSE is capable of addressing many of the properties and issues identified earlier - these results are summarised in Table 5.5

as shown below. As before the top portion of Table 5.5 refers to the capabilities identified in section 2.5, the lower portion to the issues and properties summarised in section 3.4.5. The meaning of "E", "ee" and "e" is as defined in section 4.6.

Property/Issue	Previous Work	Chapter 4	Chapter 5
Context	[41, 87]	e	**ee**
Architecture	[41, 87]	e	**ee**
Spec. Join	[39, 83]	-	**e**
Avoid Bias	[39]	e	**ee**
Model Reality	-	e	**ee**
Validation	-	e	**ee**
Simplification	[41, 87]	e	**ee**
Tool Support	[35, 87]	-	-
Right Abstraction	[41, 87]	e	**ee**
Integrates	[88]	e	**ee**
NecessaryP	[41, 83]	-	**e**
SameA1	[88]	e	**ee**
FormalA2	[41, 88]	-	**e**
IntegA3	-	e	**ee**
EnvirI1	[87, 88]	e	**ee**
RequI2	[87, 88]	e	**ee**
LateI3	[85, 88]	-	**e**
TraceI4	[85, 88]	-	**e**
IncomI5	-	-	-
ValidI6	[88]	-	**e**

Table 5.5: Evidence Status for POSE after Chapter 5

The contribution of this thesis so far to the knowledge concerning POSE and its use is summarised in Table 5.6. The "Reference" column refers to section numbers unless otherwise stated.

Property	Contribution Description	Reference
[NecessaryP]	Demonstrated support by POSE PPT process	5.1.2, 5.1.3, 5.5.3, 5.5.5
Spec. Join	Demonstrated support by POSE PPT process	5.1.2, 5.5.3, 5.5.5
Validation	*SMS* case study supports property	5.1.3, 5.5.5
[ValidI6]	*DC* and *SMC* case study support	5.1.3, 5.5.5
[LateI3]	*DC* and *SMC* case study support	5.2.2, 5.5.5
Simplification	Demonstrated support by POSE PPT process	5.1.2, 5.5.2, 5.5.5
[TraceI4]	Showed trace&iteration mitigation	5.3
[SameA1]	Showed PSA uses same model as Development	5.1.3, 5.5.5
Right Abstraction	*SMS* case study supports property	5.1.3, 5.5.1
[IntegA3]	*SMS* case study supports property	5.5.1
Integrates	*SMS* case study supports property	5.5.1
[FormalA2]	*SMS* case study support	5.5.4, 5.5.5
[EnvirI1]	*SMS* case study initial steps	5.5.1
[RequI2]	*SMC* case study initial steps	5.5.1
Architecture	*SMS* case study initial steps	5.5.1
Context	*SMS* case study supports property	5.5.1
Avoid Bias	*SMS* case study supports property	5.5.1
Model Reality	*SMS* case study supports property	5.5.1

Table 5.6: Thesis Contribution Summary after Chapter 5

Chapter 6

Further Process Improvement Using POSE

The aim of this chapter is consider how POSE can address the three items currently outstanding as indicated by Table 5.5. The first task is to demonstrate how POSE can improve support for the early application of preliminary safety analysis to provide mitigation for **[LateI3]**. The second task is to investigate what support POSE can provide for the issue **[IncomI5]** and the third is to consider **Tool Support**. The investigation will be conducted through the use of an Audio Warning System case study, which, as we will see, will also provide additional evidence for the properties and issues already addressed in earlier chapters, and further demonstrate the use of POSE in support of a formal process that is suitable for a safety critical system development.

6.1 A POSE Safety Pattern Identified: [LateI3]

Inspection of the *DC* case study of sections 4.2 – 4.3 and the *SMS* case study of section 5.5 shows a definite recurring sequence of problem transformation steps emerging for covering the early requirements engineering stages of a system develop-

ment. This sequence of events was also identified in other case study work conducted by the author at the Company, but not reported here. For example, the first problem transformation step in both the *DC* case study (PS1 on page 91) and the *SMS* case study (SPS1 on page 244) was concerned with understanding the system problem, the environment and the requirement. Further, the second problem transformation step (PS2 for the *DC* and SPS2 for the *SMS*) was concerned with identifying an appropriate solution architecture and expanding the problem context to include it. The subsequent problem transformation steps (e.g., PS3 to PS6 for the *DC* case study) were concerned with applying the PPT to simplify the problem to facilitate both the PSA and the derivation of the machine specification of the machine to be designed. This sequence of problem transformation steps can be captured by the following activities:

1. **Initial Problem Understanding:** used to capture increasing knowledge and detail in the context (i.e., the environment into which the solution will be introduced) and requirement of the problem.

2. **Solution Interpretation and Expansion:** the choice and subsequent structuring of the solution according to a candidate solution architecture. Domain knowledge and experience are used to investigate potential solution architectures (logical and/or physical), and the most promising candidate is selected to transform the problem. This involves an expansion step which requires the introduction of architectural features that are known to have a high likelihood of success.

3. **Problem Simplification:** the removal of domains by repeated application of the PPT with the goal of simplifying the problem structure to focus on the machine domain to be designed. This simplification facilitates the specification derivation and validation, and the safety analysis tasks.

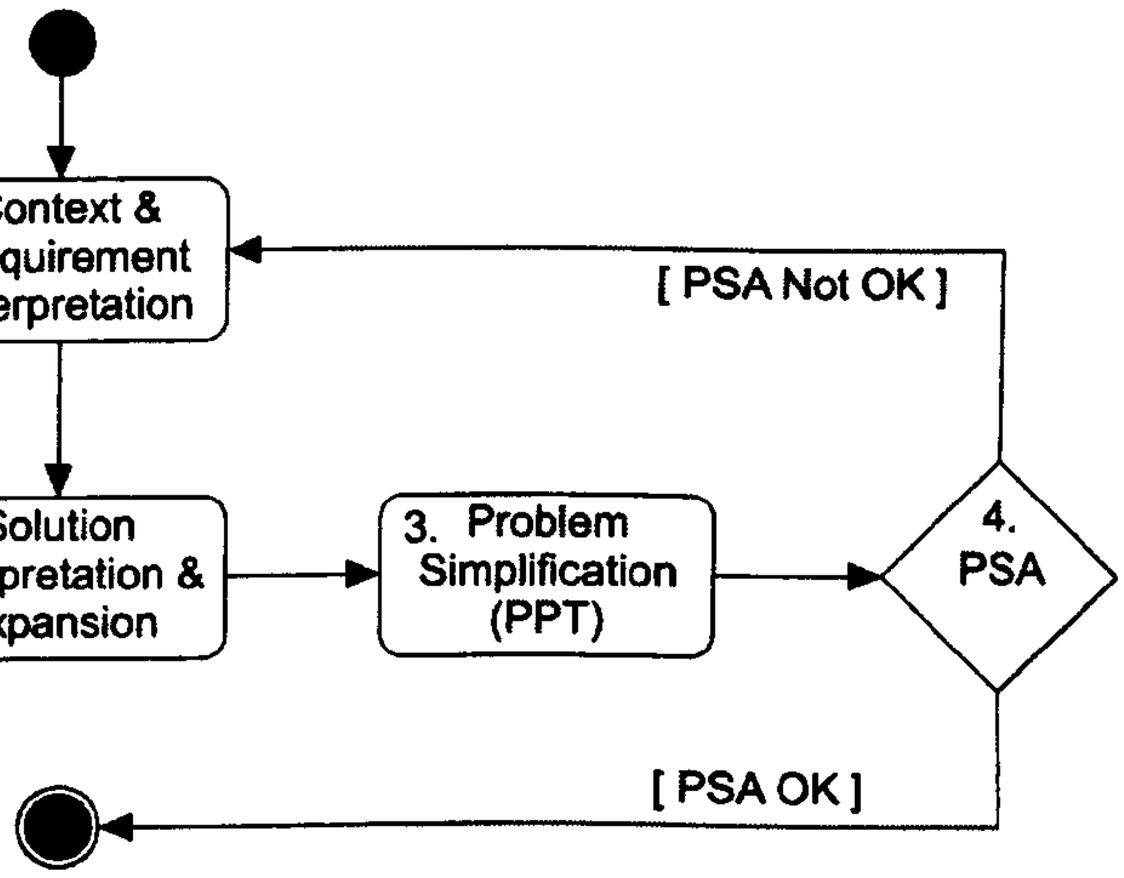

Figure 6.1: The POSE Safety Pattern

4. **PSA Feasibility Check:** the application of safety analysis techniques to the POSE model resulting from the simplification task to ensure a feasible solution architecture has been chosen.

These activities can be put together in what we define as the POSE safety pattern (Pattern, for short), which is captured by the UML activity diagram of Figure 6.1. The choice point (labelled 4) uses the outcome of the PSA to determine whether:

- PSA OK: the current architecture is viable as the basis of a solution; or
- PSA Not OK: backtracking and (re-)development of the problem (Activity 1) and/or another candidate architecture (Activity 2) is required.

The Pattern is iterative, ending when the PSA of a validated specification that satisfies its safety requirements is achieved.

The Pattern has evolved through its development but has retained the same basic form. The earliest form was presented in [87] and included an additional requirements interpretation task prior to the PSA. After further development of the POSE transformations the POSE safety pattern was modified by the removal of this additional task and the inclusion of a PSA activity that combined the problem simplification and the safety analysis tasks – this was the form used in [88]. The

combined PSA task that mixed the PPT with the PSA activity was not really satisfactory and this resulted in a further modification to the Pattern to its current form. This modification has the virtue that the POSE transformations and the PSA are distinct. That is, in the current pattern shown in Figure 6.1, Activities refer to POSE problem transformation steps, whilst the decision diamond represents the PSA.

Validation is an essential aspect of the POSE transformation process, and it is instantiated in the POSE safety pattern through the notions of justification and claim used in the problem transformation step representation (e.g. see PS1 on page 91). However, it is not explicitly represented in Figure 6.1. Validation after each step is implicit in the diagram. Ongoing work [37] has developed the process pattern further to explicitly include the validation tasks for particular problem contexts. However, the form shown in Figure 6.1 is apposite for this work to support the safety process development.

The mechanics of how to use the POSE safety pattern are represented by the diagram in Figure 6.1 and defined by the Agenda-based method description [17] (refer to section 2.3.6) shown in Table 6.1. The Validation fields in Table 6.1 are based on using IPDP, but the Description and Result fields are general. Inspection of these fields show they represent required features of any safety development process required to satisfy the safety standards and guidelines such as DS 00-56, IEC 61508 and DO-178B. Therefore the POSE safety pattern Agenda description can be readily adapted for use with other safety critical design approaches, with just the Validation fields requiring tailoring to cover the specific artefacts resulting from the actual development process used.

The POSE safety pattern was tried out on some small case study examples (not reported here) and "re-played" on the Decoy Controller and SMS case studies – refer also to Appendix H. The results obtained validated its effectiveness. The information from the IPDP sub-system theory of operation task in the Dem/Val phase is

No.	Description	Result	Validation
1.	Set up Initial Problem Understanding step. (Ref: PS1 & SPS1)	Understand basic system context, the environment domains, requirements, their phenomena and inter-relationships between them.	- Requirements confirmed by Customer - Model checked by domain experts - Justification for the transforms - Step Claims discharged
2.	Set up Solution Interpretation & Expansion step. (Ref: PS2 & SPS2)	Experience and domain knowledge used to identify a possible solution architecture. Problem expanded using this architecture.	- chosen architecture satisfies functional requirements - Justification for additional expansion discharged - Specific Claims satisfied, e.g. validate Problem is sound.
3.	Problem Simplification by successive application of the PPT problem trans. step. (Ref: PS3-6 & SPS3-4)	Domains far from the Machine are removed in turn using the PPT. Requirements correspondingly re-written to compensate. Model focuses on Machine and its Spec.	- PPT process followed & Justification - Specific Claims to support model are satisfied & valid representation. - Must be able to derive S from $W, S \vdash R$
4.	PSA of reduced model to test for feasibility. (Ref: PS6 & Section 4.5.7)	Model is feasible if safety requirements are satisfied. Otherwise iterate back to Step 1 or Step 2 as appropriate.	- Results checked by independent safety expert. - Results confirmed/accepted as part of IPDP SDR

Table 6.1: POSE Safety Pattern Agenda Description

used in the POSE safety pattern Activity 2 task to expand the architecture. The simplification (Activity 3) just uses information from the earlier steps. The latter did require clarification of some of the IPDP Dem/Val tasks to ensure that the appropriate information was always made available (ambiguity in the task description was removed). Therefore the PSA check (Activity 4) can be performed as part of the SDR work at the end of Dem/Val as desired. This fully supports the mitigation of **[LateI3]** by allowing the safety analysis (PSA) to be performed before EMD in the life cycle. It also demonstrates that the POSE safety pattern integrates well with IPDP in support of **Integrates** and **[IntegA3]**.

6.2 Remaining Issue and Property

The aim of this section is to consider what mitigation/support is or could be given to the outstanding issue of task incompleteness **[IncomI5]** and the property of **Tool Support**.

6.2.1 Support for [IncomI5]

The problem transformation steps we have defined collect together basic POSE transformations into meaningful units that achieve an identifiable development task, such as problem simplification using the PPT. The POSE safety pattern organises the steps into a repeatable process pattern that covers the tasks necessary to complete the early life cycle phases of a safety development. Following the pattern/step combination ensures that all the necessary tasks are preformed in the appropriate order, and this provides some mitigation for issue **[IncomI5]**.

6.2.2 Tool Support

A proof-of-concept tool has been developed to support POSE development written in Prolog as reported in [38], but this is rather primitive and it has not been used as part of this work. Instead, the *SMS* case study has demonstrated that POSE works well with Alloy to produce satisfactory results, and that the Alloy formal model was derived directly from the POSE problem model and its associated information. This will be investigated further in the *FAS* case study considered next. Therefore, although POSE does not provide any tool support directly, this is mitigated by the ability of using POSE models within other tools, like Alloy.

6.3 Audio Warning System Case Study

The case study concerns the design of a Failure Annunciation System (*FAS*) that is part of the warning system on a military aircraft. The *FAS* provides audio warnings to the pilot if certain critical monitored systems have failed. The case study assumes that aircraft level and system level safety analyses have been completed and have allocated safety requirements to the *FAS* - these being H1 and H2 which are defined as follows:

H1 : Inadvertent indication of the Catastrophic message.

H2 : Failure to indicate the Catastrophic message.

where the catastrophic message reports a major system failure such that the pilot is required to prepare for ejection.

These two hazards do not have the same nature: hazard H1 is particularly problematic because of the action the pilot has to take if the Catastrophic message is played. As a result, it is classified as safety critical, and assigned a target failure probability of 10^{-7}fpfh (failures per flight hour). In contrast, hazard H2, the failure to indicate a Catastrophic message, is mitigated by other, diverse, aircraft warning systems. Hence H2 is classified only as safety related, and assigned a target failure probability of 10^{-5} fpfh. The importance of HCI is increasingly recognised, but the ergonomic aspects are handled at the (higher) aircraft level and factored into the safety requirements. That is, at this level the pilot is assumed not to impact any allocated safety budget and to behave as trained for the purposes of operating the system to be developed.

6.3.1 POSE Safety Pattern: Activity 1

The POSE safety pattern, developed in section 6.1 and shown in Figure 6.1, was applied to the case study. The first activity in applying the pattern is Initial Problem Understanding and the tasks associated with this activity are represented in the problem transformation step WPS1 detailed in Appendix F.1. This captures understanding the environment context, the requirements and setting up the initial software problem via the AStruct as detailed in section 4.1.3. This work uses information from the IPDP CE and early Dem/Val phases. As with the *DC* case study the goal was to use the original requirements unmodified as far as possible to show that POSE can work with real problems, although some clarifications were incorporated to ensure that the requirements were at an appropriate level of abstraction

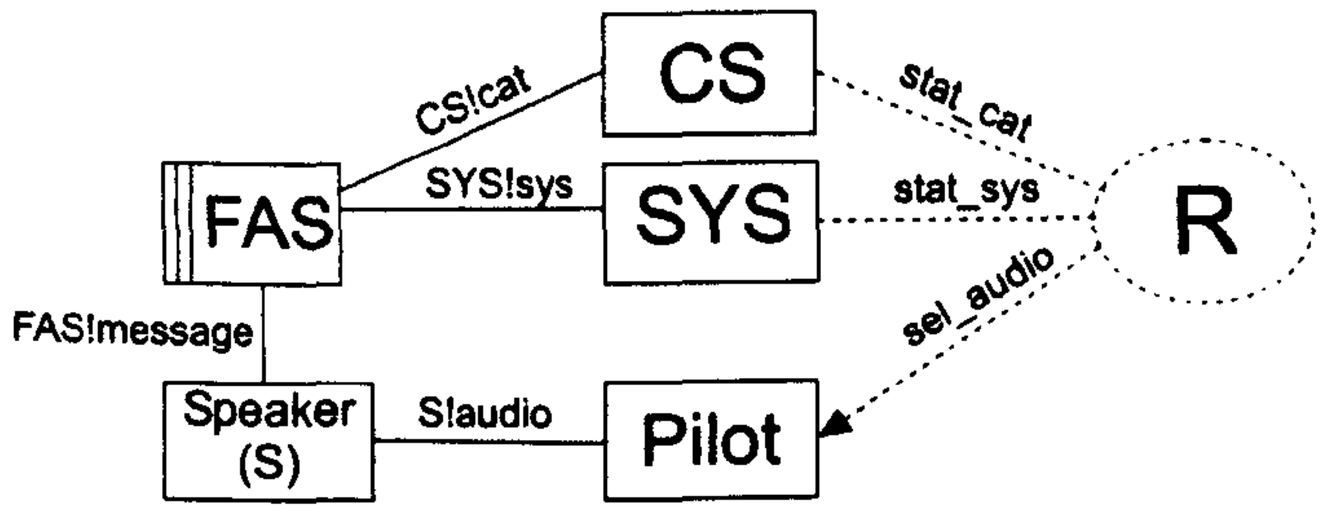

Figure 6.2: The Warning System

(refer to section 5.1.3) to facilitate the PPT application and formal validation work that follows. The Step WPS1 transforms the problem from P_{null} into $P_{Initial}$, which has the form:

$$P_{Initial}: \quad \begin{array}{l} CS(stat\_cat)^{cat},\ SYS(stat\_sys)^{sys},\ Pilot(sel\_audio)_{audio}, \\ Speaker^{audio}_{message},\ FAS^{message}_{cat,sys} \end{array} \vdash R^{sel\_audio}_{stat\_cat,stat\_sys}$$

This model can also be represented as a PF problem diagram as shown in Figure 6.2.

Appendix F.1 gives the details of the step, which indicates that all the required justifications, claims and obligations are discharged to provide the validation evidence to support the model represented by $P_{Initial}$.

6.3.2 POSE Safety Pattern: Activity 2

The second Pattern activity of Solution Interpretation and Expansion involves expanding the *FAS* to include a candidate solution system architecture and thus supports the property **Architecture** from Table 2.3. The activities associated with this task are represented in the WPS2 Step shown below and based on the PS2 Step from section 4.3. It uses information obtained from the IPDP sub-system Theory of Operation task in the Dem/Val phase.

WPS2: Application of SOLUTION INTERPRETATION AND EXPANSION *to problem* $P_{Initial}$

JUSTIFICATION J_2: The expanded *FAS* solution system architecture consists of four types of domain. The domain components in this architecture have been prototyped to show they can handle the required functionality, and the intent is to use these components in the actual design to reduce cost and risk. The initial design for the *FAS* is to make use of these off-the-shelf components for Failure Detection (the *FD* domain), for Audio Output Selection (the *AO Selector* domain) and for Audio Output Decoding (the *AO Decoder* domain), combining them together with a (still to be designed) Failure Annunciator Controller (*FA*) domain.

The *FD* receives status information from (a) the *CS* system via the discrete *cat* signal, and (b) the other *SYS* systems via *sys*. These are decoded by *FD* and sent to the *FA* via its *status* signal. The *FD* prioritises the failure data to send to the *FA* - *cat* is the highest priority.

The combined role of the *AO Selector* (*AOS*) and *AO Decoder* (*AOD*) is to output the audio signal of the message selected by the *FA*. The *AOS* is an FPGA-based device [49], containing a library of digital audio messages stored in PROM (Programmable Read-Only Memory). The *AOD* decodes the selected digital message and turns it into an audio wave for the speaker. If no failures are reported by the *FD*, then the *FA* sends no message request to the *AOS*, otherwise the *FA* sends out the highest priority message request.

The justification for this (architecture) solution expansion transformation, J_2, is that this system architecture has been successfully prototyped and has also been used on a number of other similar developments. Therefore it is known to be capable of satisfying the functional requirements.

PHENOMENA: The new phenomena introduced by the architecture are:

messdig	The selected message in digital form.
status	Combined status indication of the health of the *CS* and other *SYS* systems.
sel	The identifier of the message selected for playing to the Pilot.

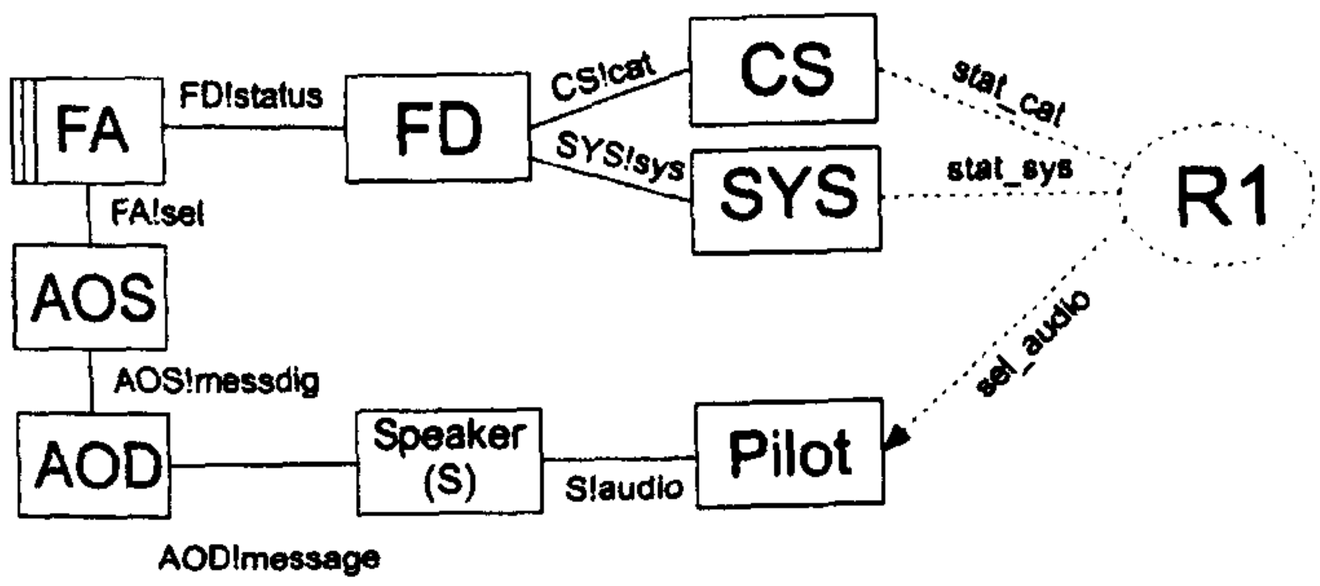

Figure 6.3: The Expanded Warning System

Given the initial representation of the problem $P_{Initial}$, the POSE expansion transformation results in the software problem becoming P_{Exp}, as shown below.

$$P_{Exp}: \quad \begin{array}{l} CS(stat\_cat)^{cat},\ SYS(stat\_sys)^{sys},\ Pilot(sel\_audio)_{audio},\ Speaker^{audio}_{message}, \\ AOS^{messdig}_{sel},\ AOD^{message}_{messdig},\ FD^{status}_{cat,sys},\ FA^{sel}_{status} \end{array} \quad \vdash \quad R1^{sel\_audio}_{stat\_cat,stat\_sys}$$

The system architecture and resulting system design is shown in PF problem diagram form in Figure 6.3.

The transformed requirement $R1$ is similar to R apart from the changes introduced by expanding the domains from FAS to FA, FD, AOS, and AOD. This only results in a change to $R1a$, where FAS is replaced by FA.

R1a When the CS or a monitored system has failed, the FA **shall control the system** to play the correct audio message to the pilot.

R1b Message levels shall be comfortably heard by the Pilot.

R1c If more than one system has failed messages shall be selected for play in the order: $stat\_cat$ fail , $stat\_sys$ fail.

R1d If no system failures are detected, then no message shall be played.

RS For hazards H1 and H2 defined above, their respective safety targets (10^{-7}fpfh and 10^{-5}fpfh) shall be satisfied.

CLAIM: *Justification for selection of the solution architecture*

ARGUMENT & EVIDENCE: This system architecture has been successfully prototyped and has also been used on a number of other similar developments. Therefore it is known to be capable of satisfying the functional requirements.

CLAIM: *The choice of candidate solution architecture exhibits sound safety engineering judgement*

ARGUMENT & EVIDENCE: The system architecture is chosen to maximise integrity by partitioning the functionality into simple, distinct blocks. The goal is to isolate any problems to specific blocks which will allow local mitigation. This strategy has proven successful on previous designs.

CLAIM: *The Speaker is reliable*

ARGUMENT & EVIDENCE: The *Speaker* is an integral part of the Pilot's headphones. These headphones have been in-service for many years and are known to be adequately reliable and give satisfactory performance.

CLAIM: *The chosen solution architecture does not prevent the satisfaction of R.* **This claim is not yet supported - deferred until PSA after PPT.**

6.3.3 POSE Safety Pattern: Activity 3

The third pattern activity is Problem Simplification using the PPT. The goal of this activity is to simplify P_{Exp} to facilitate the safety analysis (PSA) task and allow the derivation of the specification. Domains adjacent to the requirement are removed

first, following the process introduced in section 5.1.2 – in this case the first domain to be removed will be the Pilot. The step WPS3, is detailed in Appendix F.2 and is based on the PS3 Step from section 5.1.3.

The first application of the PPT transforms the software problem from P_{Exp} into P_{Red1} as follows.

$$P_{Red1}: \quad \begin{array}{l} CS(stat\_cat)^{cat},\ SYS(stat\_sys)^{sys},\ Speaker(audio)_{message}, \\ AOS_{sel}^{messdig},\ AOD_{messdig}^{message},\ FD_{cat,sys}^{status},\ FA_{status}^{sel} \end{array} \quad \vdash \quad R2_{stat\_cat,stat\_sys}^{audio}$$

At the same time the requirement is transformed from $R1$ into $R2$.

The simplification goal requires additional applications of the PPT to remove domains from the model. For brevity, these are not listed here, but are recorded in Appendix F. As the PPTs are applied, the requirements progress from $R2$ through to $R6$. As noted at the beginning of section 6.3, the *Pilot* does not impact the allocated safety budget. However, this is not the case for the AOD and CS domains removed in the problem simplification sequence, and appropriate amounts of the budget have to be allocated for them as described in Appendix F. These affects on the safety budget are reflected in **R6S**. Inspection shows that the domain removals from Appendix F are all examples of *Environment Constraint* requirements. These domain removals remove the *Speaker*, AOD, SYS and CS domains and result in the reduced software problem P_{Red} shown below.

$$P_{Red}: \quad AOS(messdig)_{sel},\ FD(cat,sys)^{status},\ FA_{status}^{sel} \quad \vdash \quad R6_{cat,sys}^{messdig}$$

This sequence of domain removals simplifies the problem structure and provides sup-

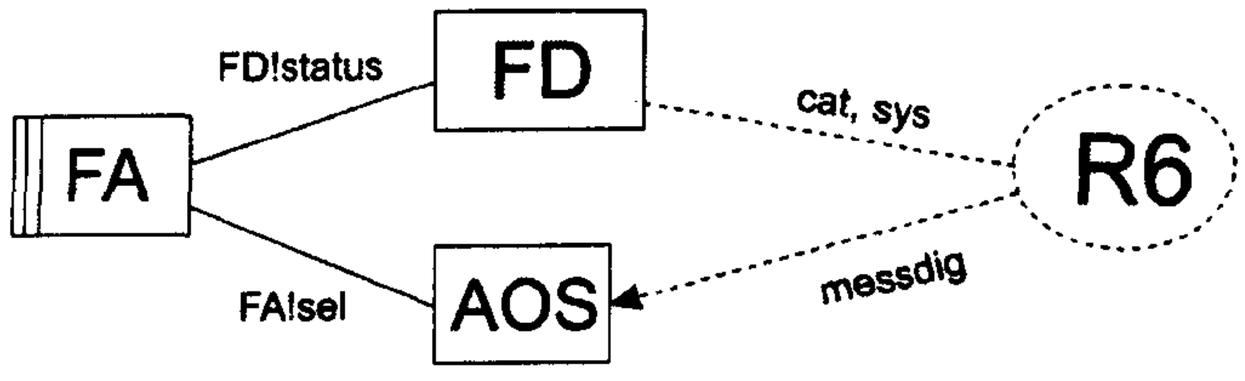

Figure 6.4: The Warning System After PPT Application

port for the property **Simplification** from Table 2.3. The requirement associated with this software problem, *R*6, is presented below.

R6a When the *CS* or a monitored system has failed, the *FA* shall control the system to generate the correct digital message stream sequence, *messdig*, at the output of *AOS*.

R6b Null

R6c If more than one system has failed messages shall be selected for play in the order: *cat* fail, *sys* fail.

R6d If no system failures are detected, then no digital message stream shall be generated.

R6S For hazards H1 & H2 defined above, their respective safety targets (10^{-7}fpfh & 10^{-5}fpfh) shall be satisfied. (Note some of H1 budget used for *CS*; some of H2 budget used for *AOD* and *CS*, but *SYS* has no impact).

The reduced Warning System is shown in PF problem diagram form in Figure 6.4.

To derive an Alloy model from P_{Red} requires reconciliation of the requirements and domain phenomena, i.e., the relationship between them needs to be established. This could be achieved by applying the PPT to remove the *AOS* and *FD* domains to allow the specification to be derived, but as in the *SMS* case study the preferred way forward (as the model is useful for the PSA) is to just apply the REQUIREMENTS

INTERPRETATION transformation to extract the relationship between the phenomena. This can be captured as the PrePSA problem transformation step below, which forms the last part of the Pattern Problem Simplification task.

The PrePSA Step manages the transformation of P_{Red} into P_{Equiv}, which can be used to directly derive the specification of the *FA* and forms the basis of the PSA work.

PrePSA: Application of REQUIREMENT INTERPRETATION *to problem* P_{Red} *to form* P_{Equiv}

CONCERNS: Reconciliation of the requirements and domain phenomena. The requirements *R*6 (associated with P_{Red}) refer to *cat*, *sys* and *messdig*, whilst the *FA* relates to *status* and *sel*. Therefore there is a need to apply REQUIREMENTS INTERPRETATION to identify the relationship between these phenomena, as in section 5.5.3.

JUSTIFICATION *J*: Inspection of the useful behaviour relation for the *AOS* and *FD* domains indicate the logical equivalence of *sel* to *messdig* and *status* to *cat* and *sys*.

This is justified by identifying the useful behaviour relations (domain contributions) of the *AOS* and *FD* domains. The *FD* domain is a reference-type, so its *effect* the health status information of *cat*, *sys* is represented by *status* which combines the health status information in priority order. The *AOS* domain is a constrain-type, so its *effect* the digital message output stream *messdig* is caused by the input phenomena *sel* as follows. The *AOS* consists of a PROM which contains all the warning messages as strings of consecutive digital bytes, as shown in Figure 6.5(a). The input phenomena *sel* indexes into the PROM to select the message to be played. The bytes from this selected message are then output in sequence as the digital message stream *messdig*. Therefore *status* relates to *cat* and *sys*, whilst *sel* relates to *messdig*. This means that the requirements *R*6 can be updated to *R*7 as follows:

R7a When the *CS* or a monitored system has failed, the *FA* shall control the system

to **select the correct message to be output.**

R7b Null

R7c If more than one system has failed messages shall be selected for play in the **correct priority order by** *status* **– catastrophic failures have the highest priority.**

R7d If no system failures are detected, then no digital message **shall be selected for output.**

R7S For hazards H1 and H2 defined above, their respective safety targets (10^{-7}fpfh and 10^{-5}fpfh) shall be satisfied. (Note some of H1 budget used for CS; some of H2 budget used for AOD and CS, but SYS has no impact).

These updates allow the problem to be transformed from P_{Red} into P_{Equiv} as follows.

$$P_{Equiv} : \ AOS(sel)_{sel},\ FD(status)^{status},\ FA^{sel}_{status} \ \vdash \ R7^{sel}_{status}$$

CLAIM: *The software problem P_{Equiv} allows the specification of FA to be derived.*

ARGUMENT & EVIDENCE: The requirement $R7$ uses the same phenomena as the domains, i.e., $R7$ is the specification. This allows the specification of FA to be derived as shown in the Alloy model development of section 6.3.4.

CLAIM: *The requirements model is a valid representation of the system*

ARGUMENT & EVIDENCE: Informally, **R7a** and the AOS contribution mean that *sel* and the AOS useful behaviour result in *messdig*, whilst **R7c** and the FD contribution mean that *status* and the FD useful behaviour result in *cat*, *sys*. This will be formally checked as part of the Alloy simulation and proof work.

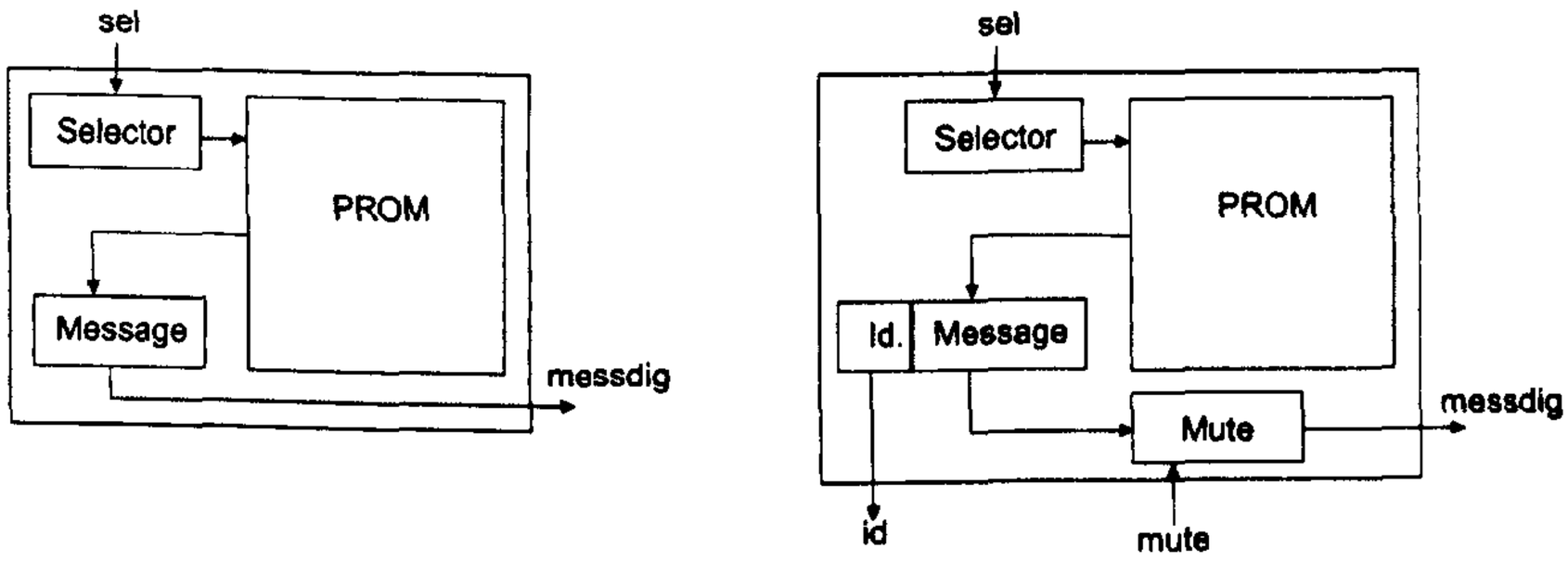

Figure 6.5: (a) Original *AOS* (b) Modified *AOS'*

6.3.4 POSE Safety Pattern: Activity 4

The PSA process developed in section 5.2 was applied to the *FAS* case study to check the feasibility of the design. The safety analysis used mathematical proof on Alloy models for the functional behaviour and a combination of FFA, HAZOPS and functional FTA to investigate the (hardware) random failure aspects.

6.3.4.1 Model Development

The first part of the PSA was to check and validate the functional behaviour of the system, and the simplified POSE problem, P_{Red}, was used to form an Alloy model of it. The model form used was a development of the style used on the *SMS* model of section 5.5.4, and is therefore based on the models given in Chapter 6 of [52]. In this case, the domains are modelled as individual abstract data types and the Alloy model forms a template which can be re-used to directly encode subsequent POSE requirements models. Note again that the same model used in the development (P_{Red}) is used as the basis for the safety analysis.

Producing the formal requirements model in Alloy has two main benefits: the requirements model can be simulated to show that it has the desired behaviour, and the individual requirements (e.g., **R6a**, **R6c**) can be encoded as predicates which can then be proved against the requirements model using Alloy's check predicate facility. The first step is to build the requirements model as discussed above, and this model is shown in full in Figure 6.6.

In the following text a reference to "#" identifies the part of Figure 6.6 being discussed. Time is modelled as a simple linear ordering using the supplied utility ordering. The interface phenomena (e.g., *sel* and *status*) are modelled as types. For example *status* defined at #1 in Figure 6.6, consists of *catnon* meaning only the *CS* system has failed, *sysnon* meaning only a *SYS* has failed, *catsys* meaning both the *CS* and *SYS* indicate failures and *non* meaning no failures detected. The *sel* (#2) and *mess* (#3) phenomena can be modelled similarly. The domains are defined as data types shown in #4 in Figure 6.6. The behaviour of these domains is defined using predicates which define the changes from the current time (t) to the next time (t'). These behaviour predicates are shown at #5 in Figure 6.6, where *FA* is represented by FABehave, *FD* by FDBehave and *AOS* by AOSBehave.

The function *extract[]* (#7) defines how the *FA* interprets the status information from the *FD*. The function *decode[]* (#8) translates the message selection given by *sel* into the appropriate message. There are two models for the *FD* behaviour. The first as shown in Figure 6.6 is the generic one as used in [52]. This model is good for the proof work, but is problematic for the simulation work since the user has little control over which trace behaviour is selected. To overcome this limitation a second model was introduced which allows control over the time range when phenomena were selected – this is an evolution of the time model used in the *SMS* case study of section 5.5.4. This second model uses the function *trange[t, lower, upper]* to set phenomena values over times ranging from **lower** to **upper**, and is shown below.

```
pred FDBehave (t,t' : Time) { trange[t, 0, 1] => FDSame[t,t']
else trange[t, 2, 4]  => (FD.status.t' = catnon )
     else trange[t, 5, 6] => (FD.status.t' = non )
          else trange[t,7,9] => (FD.status.t' = sysnon )
               else trange[t,10,11] =>  (FD.status.t' = non )
                    else trange[t,12,13] =>  (FD.status.t' = catsys )
                         else trange[t,14,15] =>  (FD.status.t' = non )
                              else trange[t,16,24] =>  (FD.status.t' = catnon )
                                   else trange[t, 25, 26] => (FD.status.t' = non )
                                        else FDSame[t,t'] }
```

The function *trange[]* (#6 in Figure 6.6) uses the fact that time is modelled as a linear ordering, so the function prevs[] can be used to effectively "count" the previous time instants. This allows the range of values to be set up as required, e.g. *trange[t,2,4]* => *FD.status.t'* = *catnon* is true if t is in the range from 2 to 4, and sets *FD.status.t'* to *catnon* over this range.

The trace behaviour (#9) is defined using the function *init[]* and the fact *traces*{} and follows the model presented in [52]. The function *init[]* places the system in a known safe state (satisfies the initialisation concern). The simulation is run by invoking the predicate show1 (#10) which runs through this trace behaviour.

6.3.4.2 Model Validation

The *SMS* case study formal modelling work showed that the simulation was quicker to set up but the exhaustive testing was time consuming. In contrast the proof took longer to set up but gave stronger results more quickly. This suggested using the simulation to set up the model, and then use proof to complete the validation and to check any safety properties. In addition, any problem areas identified by analysis of the safety requirements could be investigated using simulation. This was the approach adopted on this *FAS* case study.

The model was developed to support the PSA, so it is based on P_{Red} and its associated phenomena (and also the PF diagram shown in Figure 6.4). The problem P_{Equiv} was used to derive the specification ($R7$) of the machine to be designed, *FA*, and this was checked using simulation runs. The rest of the model (including *AOS* and *FD*) was then checked also using simulation runs. At this point there was enough confidence in the behaviour of the model developed in section 6.3.4.1 to begin proof validation.

The first proof task was to demonstrate that the model of *FA* satisfied its specification $R7$. This required the sub-parts of $R7$ to be encoded into Alloy as "Assert" statements. These were carefully reviewed to ensure they were accurate (to ensure

```
//### Problem PRed: Unmodified AOS & General FD ###
open util/ordering[Time] as Ti
sig Time {}

// ### Define Phenomena ###
sig OK {}
one sig  yes, noo extends OK {}

sig Status {} // Covers FD input cat, sys or no message
 one sig  catnon, sysnon , catsys, non extends Status {} //#1

 sig Sel {}
 one sig  m1, m2, m3, ni extends Sel {}                    //#2

 sig Mess {} // Covers AOS output messdig
 one sig  mes1, mes2, mes3, noni extends Mess {}           //#3

 // ### Define Domains ###                                  #4
 one sig FA { sel : Sel -> Time } {}
 one sig FD {status : Status -> Time} {}
 one sig AOS {mes : Mess -> Time} {}

 // ### DEFINE OPERATIONS ###                               #5
pred FABehave(t,t':Time) { FA.sel.t' = extract[FD.status.t] }

pred FDBehave (t,t':Time) {FD.status.t' = catnon or
                           FD.status.t' = catsys or
                 FD.status.t' = sysnon or FD.status.t' = non}

pred AOSBehave  (t,t': Time) { AOS.mes.t' = decode[FA.sel.t] }

 pred trange [t: Time, ts,tf: Int] {#prevs[t] >= ts &&    //#6
                                    #prevs[t] =< tf}

 fun extract[st: Status] : Sel {(st=catnon or st=catsys)  //#7
                 => m1 else  st = sysnon => m2 else ni}

 fun decode[s: Sel] : Mess { s = m1 => mes1                //#8
              else  s = m2 => mes2 else noni}

 // ### Define the Trace Model ###                          #9
pred init [t : Time] {FD.status.t  = non && FA.sel.t = ni && AOS.mes.t = noni}

fact traces { init[Ti/first[]]
  all t  : Time - Ti/last[] | let t' = Ti/next[t] |
       FDBehave[t,t'] && FABehave[t,t']  && AOSBehave[t,t'] }

// ##### PROPERTIES ##### #10
pred show1() {}
run show1 for 16 but 5 int //Set bit width to cover -16 to +15

assert R6a {all t1, t2 : Time |
 (t1 in prevs[t2] && FD.status.t1 = catnon  && t1 = prev[prev[t2]]  => AOS.mes.t2 = mes1) &&
 (t1 in prevs[t2] && FD.status.t1 = catsys  && t1 = prev[prev[t2]]  => AOS.mes.t2 = mes1) &&
 (t1 in prevs[t2] && FD.status.t1 = sysnon  && t1 = prev[prev[t2]]  => AOS.mes.t2 = mes2) &&
 (t1 in prevs[t2] && FD.status.t1 = non  &&  t1 = prev[prev[t2]] => AOS.mes.t2 = noni)
}
check R6a for 16 but 5 int
```

Figure 6.6: Audio Warning System: Original *AOS*

the model represented reality). The "Assert" statement to represent *R7a* is shown below.

```
assert R7a {all t1, t2 : Time |
 (t1 in prevs[t2] && FA.sel.t1 = m1  && t1 = prev[t2]  => AOS.mes.t2 = mes1) &&
 (t1 in prevs[t2] && FA.sel.t1 = m2  && t1 = prev[t2]  => AOS.mes.t2 = mes2) &&
 (t1 in prevs[t2] && FA.sel.t1 = ni  && t1 = prev[t2]  => AOS.mes.t2 = noni)
}
check R7a for 16 but 5 int
```

The first term in *R7a* states that if **t1** is one time unit before **t2**, and if *FA.sel* is *m1* at time **t1** then the message, *AOS.mes*, that is sent at time **t2** must be *mes1*. Now, *FA.sel.t1=m1* means that the *FD* has reported a *cat* failure to the *FA* at time **t1**. From *R6a*, this requires the *FA* to select the correct message for output (via *AOS*). Inspection of the Alloy model at #7 indicates that *m1* corresponds to *cat* being detected, and this in turn corresponds to *mes1* being sent to the *AOS* (via #8). Therefore, the model selects *mes1* in response to a *cat* failure being reported (*catnon* or *catsys)*, it selects *mes2* if *sys* (*sysnon*) is reported and it selects nothing (*noni*) if no failures (*ni*) are reported. This agrees with the requirement *R7a* and was validated by the proof of *assert R7a*.

A similar "Assert" was developed for *R7c* involving combinations of *FD.status.t1* and *FA.sel.t2*, and this was also validated by proof. These two proofs also covered *R7d*.

The second proof task was to validate that the PSA model represented by P_{Red} satisfied its requirement, *R6*. This covers the validation that the specification, *R7*, and the useful behaviours of the domains *AOS* and *FD* satisfy the requirement *R6*. The Alloy "Assert" statement to cover *R6a* is shown below.

```
assert R6a {all t1, t2 : Time |
 (t1 in prevs[t2] && FD.status.t1 = catnon  && t1 = prev[prev[t2]]  => AOS.mes.t2 = mes1) &&
 (t1 in prevs[t2] && FD.status.t1 = catsys  && t1 = prev[prev[t2]]  => AOS.mes.t2 = mes1) &&
 (t1 in prevs[t2] && FD.status.t1 = sysnon  && t1 = prev[prev[t2]]  => AOS.mes.t2 = mes2) &&
 (t1 in prevs[t2] && FD.status.t1 = non  &&  t1 = prev[prev[t2]] => AOS.mes.t2 = noni)
}
```

```
check R6a for 16 but 5 int
```

The first term in *R6a* states that if **t1** is two time units before **t2**, and if *FD.status* is *catnon* at time **t1** then the message, *AOS.mes*, that is sent at time **t2** must be *mes1*. Now, *FD.status.t1=catnon* means that the *CS* has reported a *cat* failure to the *FD* at time **t1**. From *R6a*, this requires the *FA* to select the correct message for output (via *AOS*). Inspection of the Alloy model at #7 indicates that *m1* corresponds to *cat* being detected, and this in turn corresponds to *mes1* being sent to the *AOS* (via #8). Therefore, the model selects *mes1* in response to a *cat* failure being reported (*catnon* or *catsys)*, it selects *mes2* if *sys* (*sysnon*) is reported and it selects nothing (*noni*) if no failures (*ni*) are reported. This agrees with the requirement *R6a* and is validated by the proof of *assert R6a*. It is worth noting that *R6a* also validates requirement *R6c* concerning message priorities and requirement *R6d* concerning selecting no messages if no failures are detected.

In summary, this work has validated that the machine specification represented by *R7* satisfies its requirement represented by *R6*.

The functional safety requirements of the system represented by problem P_{Red} are covered by the sub-parts *R6a* to *R6d* of *R6*; the non-functional safety aspects are represented by *R6S*. Therefore the validation proof of "Assert R6a" also validates that the functional safety aspects of the PSA are satisfied – so no specific safety property proof work was required. In addition, further simulation work was undertaken to check the behaviour using intermittent *cat* health problems to confirm the model behaved correctly in the presence of rapid input changes. The results were satisfactory, and supported the proof work.

The second part of the PSA was to check the random failure characteristics of the proposed system. The FFA and HAZOPS were applied as described in the improved non-formal safety analysis of section 5.2.1.1. Functional FTA was then used to analyse if the events identified by the FFA and HAZOPS satisfy the targets

Table 6.2: *AOS* Safety Analysis Results

Id.	Failure Mode	Haz.
F1	Plays no messages - no Cat fail when required.	H2
F2	Plays Cat fail message too late.	H2
F3	Plays wrong message - inadvertent Cat fail.	H1
F4	Plays wrong message - no Cat fail when required.	H2
F5	Plays too loud - Pilot switches system off.	H2
F6	Plays too softly - Cat fail not heard.	H2

contained in *R6S*. For reasons of brevity we summarise only the main elements of the *AOS* analysis to demonstrate the process followed. The significant results from applying the FFA and HAZOPS analysis to the *AOS* are shown in Table 6.2, where each functional failure is associated with the hazard it excites.

FTA, applied to the original *AOS* system architecture (refer back to Figure 6.5(a)), with F1 to F6 as the top events, was used to establish if the architecture can satisfy its targets. F3 is dominated by the known failure rate of the FPGA that would implement the *AOS* functionality. This indicates that the failure rate for F3 is $3x10^{-7}$ fpfh [1] which, therefore, does not satisfy the target for H1. Therefore the random failure part of the PSA has identified problems with the design. The result is that the FPGA *AOS* component will need to be re-thought in order to meet all safety requirements.

The PSA as recorded above was completed after the Pattern Activity 3 Problem Simplification, which in turn was completed at the end of the IPDP Dem/Val phase when all the necessary information was available. This meant the PSA results were used to support the SDR.

[1] The manufacturers specification for the FPGA indicates a gate failure rate of $3x10^{-7}$ per hour of operation. A pessimistic view is adopted that a single gate failure will cause the effect of concern, F3.

6.3.5 POSE Safety Pattern: Backtracking

The POSE development sequence so far is captured by Figure 6.7 as a problem transformation graph. This sequence is notable in that, from the PSA, P_{Red} has no solution, so we must backtrack the development to the point at which the system architecture was introduced, and continue afresh. The architecture was introduced as part of WPS2 to form problem P_{Exp}. Therefore, from the diagram, it can be seen there is a need to backtrack to the end of WPS1, after problem $P_{Initial}$.

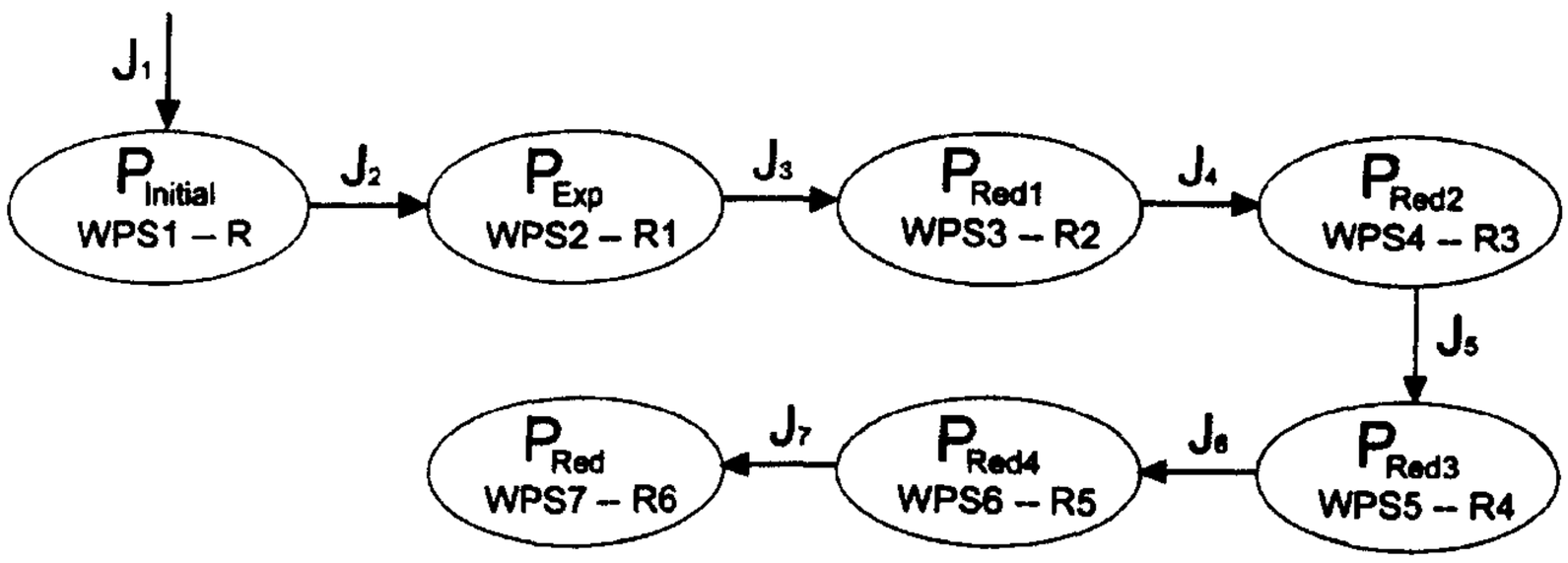

Figure 6.7: The Audio Warning Problem

The PSA problem concerns the expected reliability of the *AOS* FPGA and the fact that if it goes wrong then there is nothing the *FA* can do about it. A number of solution candidates were considered and the selected solution candidate involved including a derived requirement, *DR*, as follows:

> Each message will have a unique identifier and if the selected message identifier does not correspond to that of the currently playing message, then audio must be inhibited, audio otherwise being allowed.

To implement this derived requirement it is necessary to update the functionality of the *FA* and the *AOS* domains, and that these domains share the appropriate information concerning messages, message identifiers and the need to mute messages. This is another example of a *codesign* problem as outlined in [41].

Inspection of P_{Red} shows that the available phenomena are *messdig*, *sel*, *status*, *cat* and *sys*. None of these are suitable carriers for the required information to

support the derived requirement. That is, the information required to implement the derived requirement is not shared between the pertinent domains – this is an example of the *Unshared Information* non-feasible requirement identified in section 2.4.

Therefore to implement *DR* it is necessary to introduce additional shared phenomena between the *FA* and *AOS* domains and to appropriately update the functionality of these domains. This results in a modified FPGA *AOS* component that is developed to: include sending back the message identifier of the currently playing message to the *FA* in a status message; and add a mute input to the *FA* control that allows the *FA* to mute audio output if the message identifier does not tally with the required message to be played. The changes to the AO Selector (*AOS'*) are shown in Figure 6.5(b).

The information gained through WPS1 is re-used, the change occurs at the expansion stage in WPS2 where the additional phenomena *id*, *mute* and the derived requirement *DR* are introduced into the problem. These changes are captured by transforming WPS2 into WPS2'. The fact that much of WPS2 is re-used can be seen from inspection of the WPS2' step shown below.

WPS2': Application of SOLUTION INTERPRETATION AND EXPANSION *to problem* $P_{Initial}$

JUSTIFICATION J_2': The justification description is same as that in WPS2 (refer to page 163) together with the description of the additional phenomena *id*, *mute*, and the derived requirement *DR* given above.

The justification (J_2') for the revised system architecture has two parts and is that: (a) this basic architecture structure has been successfully prototyped and has also been used on a number of other similar developments. Therefore it is known to be capable of satisfying the functional requirements. (b) "the original POSE development sequence was infeasible because it did not satisfy the safety requirements; the revised architec-

ture can satisfy both the functional and safety requirements". That is, the revised justification includes the original pruned part of the sequence and its shortcomings (the issue identified with F3 by the PSA of the original development sequence shown in Figure 6.7), as a rationale for selecting the revised architecture that now includes *id*, *mute*, and *DR*.

PHENOMENA: The phenomena introduced by the architecture are as in WPS2 with the further addition of *id* and *mute*.

Given the initial representation of the problem $P_{Initial}$, the POSE expansion transformation results in the software problem becoming $P_{Exp'}$, as shown below.

$$P_{Exp'}: \quad \begin{array}{l} CS(stat\_cat)^{cat},\ SYS(stat\_sys)^{sys},\ Pilot(sel\_audio)_{audio},\ Speaker^{audio}_{message}, \\ AOS'^{messdig,id}_{sel,mute},\ AOD^{message}_{messdig},\ FD^{status}_{cat,sys},\ FA^{sel,mute}_{status,id} \end{array} \vdash R1'^{sel\_audio,mute}_{stat\_cat,stat\_sys,id}$$

The transformed requirements $R1'$ are the same as $R1$ apart from the addition of DR.

The claim/argument combinations same as WPS2 on page 163, apart from addition of the following claim.

CLAIM: *Requirements R1 with DR are consistent*

ARGUMENT & EVIDENCE: Comparison of DR with $R1$ indicates potential issue with **R1a** concerning the playing of warning messages, others are OK. Defer until section 6.3.7.

The revised WPS2' step, is essentially WPS2 with the changes associated with *id*, *mute* and *DR* added. This includes the additional claim/argument concerned with establishing the consistency of the revised requirements of $R1$ with DR, the

resolution of which is deferred until section 6.3.7. The addition and capture of this important claim/argument provides further evidence to mitigate **[IncomI5]**.

The next task was to revisit the WPS3 step (refer to page 269). Inspection of the transformations involved with WPS3 (*Pilot* domain removal) reveal that the addition of *id*, *mute* and *DR* do not affect the transformation. That is, technically WPS3 and WPS3' are the same apart from *DR* added to *R*2 to form *R*2′ and this *R*2′, along with *id* and *mute*, added to the appropriate parts of P_{Red1} to form $P_{Red1'}$. This follows because although **R2a** is affected by the domain removal, it is only changed with respect to the original phenomena. The change does not involve the new phenomena *id* and *mute*, nor does it impact the behaviour defined by *DR*.

A similar argument can be applied to the remaining domain removals which are not impacted by the addition of *id*, *mute* and *DR*. Therefore the POSE transformation sequence outlined in section 6.3.3 and shown in Appendix F was repeated and the revised justifications map directly to J_3 to J_7. This resulted in the reduced, simplified, software problem of P_{Red} given by:

$$P'_{Red}: \; FD(cat, sys)^{status}, \; AOS(messdig)'^{id}_{sel,mute}, \; FA^{sel,mute}_{status,id} \vdash R6'^{messdig,mute}_{cat,sys,id}$$

The corresponding PF problem diagram form is shown in Figure 6.8.

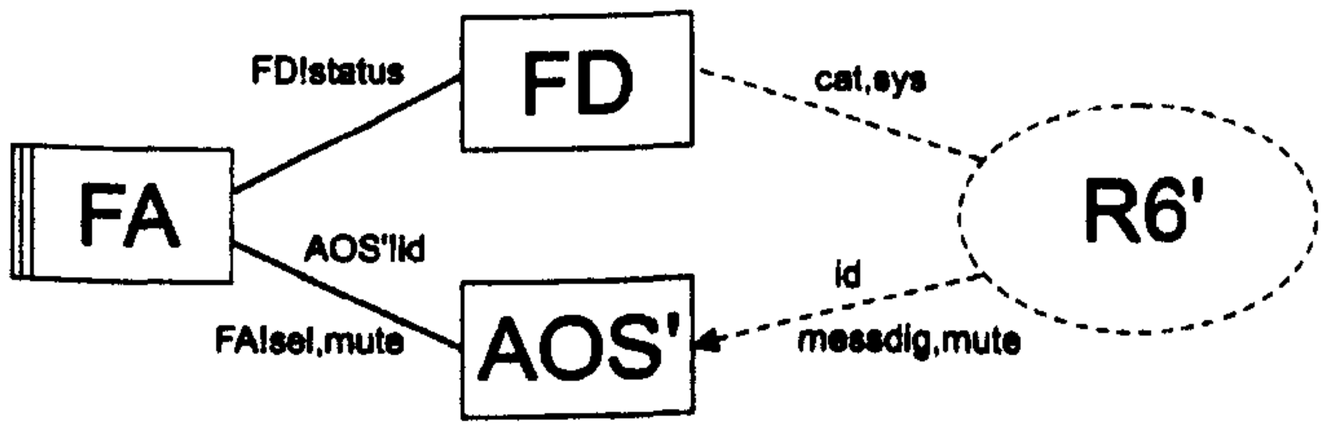

Figure 6.8: The Modified Audio Warning System, P'_{Red}

PrePSA' will also have a similar form to PrePSA, but is affected by the phenomena *id*, *mute* and derived requirement *DR*. This resulted in P'_{Equiv} shown below.

$$P'_{Equiv}: \; FD(status)^{status},\; AOS(sel)'^{id}_{sel,mute},\; FA^{sel,mute}_{status,id} \vdash R7'^{sel,mute}_{status,id}$$

6.3.6 PSA Applied to Modified Model

As before the feasibility of P'_{Red} is checked by applying the PSA tasks to investigate the functional and random failure behaviour of the revised model. The functional behaviour was validated using an Alloy model derived directly from the POSE model structure and using the POSE domains and phenomena as its basis. This was similar to the earlier model, but included the behaviour of the additional *mute* and *id* phenomena. This model is shown in full in Appendix F.7. The main changes occur in the definitions of the abstract data types for *FA* and *AOS'* as shown below.

```
// Define the single FA State - sel and mute
one sig FA { sel : Sel -> Time, mute : OK -> Time } {}

// Define the single FD State domain - status
one sig FD {status : Status -> Time} {}

// Define the single AOS' State domain - messdig and id
one sig AOS {mes : Mess -> Time, id : OK -> Time} {}
```

The behaviour of *FA* (*FABehave*) and *AOS'* (*AOS'Behave*) were modified to include the *mute* and *id* phenomena as shown below.

```
//  FABehave models the behaviour of the safety critical controller.
pred FABehave(t,t' : Time) { FA.sel.t' = extract[FD.status.t] &&
                                        FA.mute.t' = checkid[t,t'] }
```

```
.........

// AOSBehave models the behaviour of the AOS' domain. If mute is
// active then the message is set to noni (no message).
pred AOSBehave  (t,t' : Time) { AOS.mes.t' = (FA.mute.t = noo =>
                                              decode[FA.sel.t] else noni) &&
                                AOS.id.t' = setid[t,t']}

.........
// Fun checkid confirms message id corresponds to selected message #11
fun checkid[t,t' : Time]: OK {AOS.id.t = yes => noo else yes}
```

The function *checkid[]* returns **noo** if the selected message agrees with the id returned from the *AOS'*, and the audio is not muted. If the result is **yes**, then audio is muted as required.

The analysis approach adopted the same strategy introduced in section 6.3.4.2 which used simulation to set up the formal model, then proof to validate the specification and the requirement, proof to validate any safety properties (none in this case, as the requirement covered the safety properties), with the option of further simulation to explore any specific problem behaviours such as rapidly changing inputs. In this case, the behaviour of the system was simulated over a wide range of input combinations using the modified input value-time trace model (introduced on page 144) to control the simulation timing. The simulation explored the expected behaviour of the model, especially the operation associated with the *mute* and *id* phenomena. The results indicated that the requirements specification model behaved as intended. Next the proof work on the requirements was undertaken, and the full results are considered in section 6.3.7.

The second part of the PSA checked the random failure performance of the revised system. Note that DR also introduces additional failure modes. For example, *mute* failing on is a form of F6 (see Table 6.2). Table 6.3 shows the results of repeating the FFA/HAZOPS analysis, collating results and then performing the

Table 6.3: Collated PSA on *AOS′* Results

Failure Mode	**Failure Prob.**	**Haz. Id.**
Collated Failure Mode FTA Haz.	10^{-12} (10^{-7})	H1 & H2
Wrong message played, correct id	10^{-13} (10^{-7})	H1 & H2
Correct message, wrong or no id.	10^{-7} (10^{-5})	H2
No message played: mute fails/too soft	10^{-6} (10^{-5})	H2
Message played too loud	10^{-6} (10^{-5})	H2

FTA with the new system architecture. The Target - shown in brackets after the FTA calculation result - is the most severe probability applicable (e.g., the H1 target, 10^{-7} fpfh, used for first two terms).

The worst effect of "Message played too loud" is considered to be that the pilot switches the Audio Warning System, *FAS*, off to avoid the distraction - this equates to hazard H2, so the H2 target is used. Mitigation for this failure mode is provided by the Volume Limiter circuitry which is part of the *AOD* (refer to WPS5 on page 274). Inspection of the analysis results in Table 6.3 indicates that the modified *AOS′* architecture is adequate for this aspect of the safety targets, and therefore it is valid to use it to continue with the development.

A similar analysis was applied to the *FD* domain and satisfactory results were obtained. Therefore, the modified Audio Warning System architecture can be argued not to prevent satisfaction of both the functional and safety target requirements, and hence is a suitable basis for the remainder of the development process, that is, designing the *FA*. However, this is only valid if the requirements are shown to be consistent.

6.3.7 PSA Requirements Validation

The next step was to try to formally prove that the revised model satisfied its specification given by *R7′*, its requirement given by *R6′* (including its functional safety requirement) and also to cover the consistency of the requirement. The check for *R6′a* produced a counter example (i.e., the proof failed). The fact that the failure

was expected follows from inspection of DR, and $R6'a$ shown below:

R6'a When the *CS* or a monitored system has failed, the *FA* shall control the system to generate the correct digital message stream sequence, ***messdig***, at the output of AOS'.

$R6'a$ requires the correct operation of the selection mechanism, whilst DR ensures safe operation by muting audio if there is a discrepancy between the selection and the message output functions. Therefore, when the $R6'a$ predicate is applied to the modified system architecture it produces a counterexample at the point where a message should be output but the *mute* is in operation due to an identified problem with the message *id*. The resolution requires $R6'a$ to take due account of the derived requirement DR. This can be achieved by modifying $R6'a$ to become:

R6'aM When the *CS* or a monitored system has failed, the *FA* shall control the system to generate the correct digital message stream sequence, ***messdig***, at the output of AOS', except when a discrepancy is identified with the message identifier.

With this modification $R6'aM$ and DR are consistent. This is demonstrated by proving that the modified predicate corresponding to $R6'aM$ does not produce a counterexample. For example, the *assert R6aM* fragment shown below covers the "positive" behaviour showing that if there is no problem with the message identifier *id*, then *mute* is off (**noo**) and the systems outputs the correct message.

```
assert R6aM {all t1, t2, t3 : Time | (t3 = prev[t2] &&
 FA.mute.t1 = noo && FA.mute.t3 = noo && FA.mute.t2 = noo &&
 t1 in prevs[t2] && FD.status.t1 = catnon  &&
 t1 = prev[prev[t2]] => AOS.mes.t2 = mes1)
&&
### Similar terms for catsys, sysnon and non ### }
```

The proof work was extended to the other aspects of the requirements to demonstrate that the requirements model satisfied them, and that they were consistent.

This validated that the functional requirements $R6'a$, $R6'c$, $R6'd$ and DR of $R6'$ were satisfied, demonstrating that the revised model satisfies it functional behaviour requirements.

After this PSA application it has been demonstrated that the specification ($R7'$) satisfies its requirements, including its safety requirements, and the requirements are known to be consistent.

6.4 Discussion

The POSE safety pattern was identified and introduced to manage the early phases of a safety development process in section 6.1. The Audio Warning System case study of section 6.3 demonstrated the successful use of the pattern on a realistic example, and also showed that it was capable of supporting the PSA as part of the IPDP SDR in section 6.3.4. The PSA at SDR supports the goal of achieving an appropriately early safety analysis which provides strong evidence to mitigate **[LateI3]**. Further, the pattern diagram of Figure 6.1 and its associated Agenda description of Table 6.1 were found to be useful ways of representing what needs to be done to successfully apply the POSE safety pattern.

The *FAS* case study PSA work reported in section 6.3.4, section 6.3.6 and section 6.3.7 also demonstrated the successful use of the safety analysis approach introduced in section 5.2 – providing further mitigation for **[LateI3]** with respect to its aspects concerned with effective analysis.

The formal modelling work discussed in section 6.3.4 and section 6.3.6 further demonstrated that POSE integrates well with Alloy. In particular the Alloy formal model can be derived directly from the POSE software problem using a systematic process. The validation followed the sequence of validating the specification, then the requirement and then considering the functional safety requirements. In the case study these were included in the requirement, but the formal model could be used

to investigate and prove formal safety properties. Further, this work also showed the advantages of using a four-part process in applying the formal modelling:

1. use simulation to develop the model and gain confidence it has the required behaviour;

2. perform formal proof to validate the model;

3. use the model to prove safety properties (if necessary)

4. investigate any specific problem areas using simulation.

This process is efficient because the more expensive proof work is only undertaken once simulation has provided confidence in the efficacy of the model. The work also showed how the Alloy model could be derived directly from the POSE problem model structure using a systematic, repeatable process. In fact this process has been used subsequently at the Company to develop a large number of Alloy models to provide formal validation evidence in support of a number of POSE safety pattern applications. The results obtained from applying this formal analysis shows that POSE and Alloy are a good, effective combination for performing this early life cycle phase work since they: provide an effective, successful safety analysis supporting the mitigation of **[LateI3]**; use the same model as the development, supporting property **[SameA1]**; use a formal process satisfying property **[FormalA2]** and provide strong validation evidence in mitigation of **[ValidI6]** and the property **Validation**.

The *FAS* case study also demonstrated how the POSE problem transformation graph (e.g., see Figure 6.7) can be used to represent the sequence of problem transformation steps involved in transforming a complex problem into a simpler one for analysis, and how this can be used to assist with backtracking, providing support for property **[TraceI4]**. This simplification using the PPT reported in section 6.3.3 and section 6.3.5 provides support for property **Simplification**.

The case study did not demonstrate tool support for POSE, but it did provide further evidence that POSE integrates well with Alloy to provide a powerful development toolset, providing further evidence for property **Tool Support**. The case study also demonstrated that the POSE safety pattern integrates well within IPDP (see section 6.3.1 and section 6.3.2) in support of property **[IntegA3]** and property **Integrates**.

Following the POSE safety pattern means that Activity 1, based on the Initial Problem step, is completed before starting Activity 2 (Solution Interpretation and Expansion step), which ensures that a comprehensive understanding of the problem space (environment context and requirement) is obtained before starting to develop a solution architecture. The approach used for the *FAS* case study in section 6.3.1 was similar to the approach used for the *DC* case study which is detailed in section 4.1.2.1, section 4.1.3.1, section 4.1.3.2 and section 4.2. This provides strong evidence for the mitigation of the first two issues of understanding the environment context (**[EnvirI1]**) and requirements acquisition (**[RequI2]**). It also strongly supports property **Context**.

The *FAS* case study in section 6.3.1 used the same approach as the *DC* case study reported in section 4.1.2.2, section 4.1.3.2, and section 4.1.3.3. Therefore, it also provides evidence to support the property **Avoid Bias**. Similarly, the *FAS* case study in section 6.3.1 used the same approach as the *DC* case study reported in section 4.1.2.2 and section 4.1.3.1, thus also providing support for property **Model Reality**. The *FAS* case study in section 6.3.2 covering Activity 2 of the pattern, used the same approach as the *DC* case study reported in section 4.3 and this supports the **Architecture** property.

The case study work also provided support for the mitigation of **[IncomI5]** concerning task incompleteness with respect to the use of the POSE safety pattern and Problem Transformation steps. The *FAS* case study formal validation work, reported in section 6.3.4.2 and section 6.3.7, directly supports the mitigation of **[Re-**

quI2] concerning sound, consistent and complete requirements as well as [**ValidI6**] and property **Validation**. In addition, the derivation of a validated formal specification provides support for property **Spec. Join**. Care was taken in the *FAS* case study work to ensure that the requirements were at an appropriate level of abstraction (section 6.3.1) and the successful PSA and validation work reported in sections 6.3.4–6.3.7 provide support for property **Right Abstraction**.

The successful application of the POSE safety pattern in conjunction with the formal safety analysis and validation provide strong evidence concerning the feasibility of the requirements model and specification that results from the work.

6.5 Chapter Summary

The primary goal of this chapter was to introduce the POSE safety pattern and through the *FAS* case study demonstrate how it can be used to derive a specification that is capable of satisfying its requirement – including the safety requirements allocated to it. The case study also demonstrated that the POSE safety pattern could be used as part of the IPDP to mitigate the issue [**LateI3**]. In addition this chapter discussed mitigation and support for the other two outstanding items – as shown in the Chapter 5 column of Table 6.4 – namely [**IncomI5**] and **Tool Support**.

As in previous chapters, the top portion of Table 6.4 refers to the capabilities identified in Table 2.3. The *FAS* case study work has provided supporting evidence for the Context, Architecture, Avoid Bias, Model Reality, Validation, Integrates, Spec. Join, Simplification and Right Abstraction properties identified in Table 2.3. The case study also shows that POSE works well with Alloy, providing some indirect supporting evidence for Tool Support. The meaning of "e", "ee", "E" etc. is as defined in section 4.6.

The lower portion of Table 6.4 demonstrates that the case studies (*DC*, *SMS* and

Property/Issue	Previous Work	Chapter 4	Chapter 5	Chapter 6
Context	[41, 87]	e	ee	**E**
Architecture	[41, 87]	e	ee	**E**
Spec. Join	[39, 83]	-	e	**ee**
Avoid Bias	[39]	e	ee	**E**
Model Reality	-	e	ee	**E**
Validation	-	e	ee	**E**
Simplification	[41, 87]	e	ee	**E**
Tool Support	[35, 87]	-	-	**e**
Right Abstraction	[41, 87]	e	ee	**E**
Integrates	[88]	e	ee	**E**
NecessaryP	[41, 83]	-	e	**ee**
SameA1	[88]	e	ee	**E**
FormalA2	[41, 88]	-	e	**ee**
IntegA3	-	e	ee	**E**
EnvirI1	[87, 88]	e	ee	**E**
RequI2	[87, 88]	e	ee	**E**
LateI3	[85, 88]	-	e	**ee**
TraceI4	[85, 88]	-	e	**ee**
IncomI5	-	-	-	**e**
ValidI6	[88]	-	e	**ee**

Table 6.4: Evidence Status for POSE after Chapter 6

FAS) have provided evidence to satisfy the properties **[NecessaryP]**, **[SameA1]**, **[FormalA2]** and **[IntegA3]** identified in section 3.4.5. Further, these case studies have also provided evidence to mitigate the issues **[EnvirI1]**, **[RequI2]**, **[LateI3]**, **[TraceI4]**, **[IncomI5]** and **[ValidI6]** also from section 3.4.5.

In conclusion the *FAS* case study demonstrates that the POSE safety pattern, incorporating the other POSE extensions described in this chapter and in Chapter 5, can support a formal development process that is suitable for safety critical system development.

The contribution of this thesis to the knowledge concerning POSE and its use is summarised in Table 6.5. The "Reference" column refers to section numbers unless otherwise stated, and pattern refers to the POSE safety pattern introduced in section 6.1.

Property	Contribution Description	Reference
POSE Safety Pattern	Pattern developed by this work provides a structure that supports safety development	6.1 6.3
[LateI3]	Pattern supports mitigation of issue	6.1, 6.3.4
[ValidI6]	Pattern mitigates validation to issue	6.3.4, 6.3.7
[EnvirI1]	Pattern Activity1 mitigates issue	6.3.1
[RequI2]	Pattern Activity1 mitigates issue	6.3.1
Context	Pattern Activity1 supports property	6.3.1
Avoid Bias	Pattern Activity1 supports property	6.3.1
Model Reality	Pattern Activity1 supports property	6.3.1
Validation	Pattern Activity1 supports property	6.3.1
Architecture	Pattern Activity2 supports property	6.3.2
[NecessaryP]	Pattern Activity3/4 supports property	6.3.3, 6.3.4, 6.3.6, 6.3.7
Spec. Join	Pattern Activity3/4 supports property	6.3.3, 6.3.4, 6.3.6, 6.3.7
Simplification	Pattern Activity3 supports property	6.3.3, 6.3.5
[SameA1]	Pattern Activity4 supports property	6.3.4
[FormalA2]	Pattern Activity4 supports property	6.3.4, 6.3.6, 6.3.7
[TraceI4]	Pattern Backtracking mitigates issue.	6.3.5
[IncomI5]	Pattern activities mitigate issue.	6.2.1, 6.3.2, 6.3.5
[IntegA3]	The *FAS* analysis provides evidence	6.4
Right Abstraction	The *FAS* analysis provides evidence	6.4
Integrates	The *FAS* analysis provides evidence	6.4
Tool Support	The *FAS* analysis provides evidence	6.2.2, 6.4

Table 6.5: Thesis Contribution Summary: Chapter 6

Chapter 7

Discussion and Conclusions

The aim and objectives of this work were discussed in section 1.2 and concern the following. The overall aim of this work is to show that the early phases of a development process suitable for embedded safety systems development can be enhanced such that it allows a requirement model to be produced early in the development process, a specification to be derived from this model, and the safety feasibility of this model to be evaluated. The end result is a model that is known to be able to satisfy its safety requirements and thus forms a good basis for the rest of the development.

The overall aim was addressed by evaluating POSE, and the extensions introduced by this work, against two objectives. The first involved considering whether the extended POSE could address the ten SSSD approach properties identified in the literature review of Chapter 2 and summarised in Table 2.3. This first objective is discussed in section 7.1.

The second objective was to demonstrate that the extended POSE could be used to improve the early phases of an existing safety development process (IPDP) by evaluating it against a further four properties and six issues summarised in section 3.4.5. These latter properties and issues were targeted at improving the existing IPDP which is used to develop the type of embedded avionics safety systems that

are of interest to this work. This second objective is considered in section 7.2, with additional supporting evidence from an industrial case study discussed in section 7.3.

The overall conclusion of this work is given in section 7.6.

7.1 Investigation of the ten SSSD Approach Properties

The final section 2.5 of Chapter 2 defined ten desirable properties (see Table 2.3) that an SSSD approach targeted at the early phases of the development of real-time safety systems should possess. The extent to which POSE satisfied these properties was evaluated in Chapter 4, Chapter 5 and Chapter 6 using existing published work, analysis and case study evidence as appropriate. How POSE addressed each of these ten properties will now be summarised in turn. In the following, step will be used to indicate the problem transformation step concept introduced in section 4.1.4.2, pattern will be used to indicate the POSE safety pattern introduced in section 6.1 and the SSSD approaches are those reviewed in section 2.3.

7.1.1 The property Context

The property **Context** is concerned with ensuring that an approach is capable of acquiring information that supports a comprehensive understanding of the system environment context which can adequately support the system development activities. This property is related to the concept of understanding the problem before attempting the solution and is therefore a fundamental aspect of the problem oriented approaches like POSE. It was identified from the Parnas 4-variable based SSSD approaches (Parnas Table, SCR, SpecTRM, AMBERS, PFS), and is also an important part of AdLS, and KAOS. It also has much support in the published literature [54, 82, 121, 127].

The concept of "context" is a key component of POSE and one way this is indicated is by the existence of a CONTEXT INTERPRETATION transformation, and which is carried through into its use in the Initial Problem step and as Activity 1 in the pattern. The first part of the *DC* case study (section 4.1.2.1) was concerned with collecting information about the environment context and this information was used in the application of the CONTEXT INTERPRETATION transformation in section 4.1.3.1 to introduce the context into the initial problem as part of the Initial Problem step detailed in section 4.2. The Initial Problem step was also used in the *SMS* (section 5.5.1) and *FAS* (section 6.3.1) case studies, so these provide further evidence to support the property. Further, the successful application of the PPT relies on an accurate understanding of the useful behaviour relation of the domain to be progressed (part of the problem context), since this becomes the assumption which is used as part of the validation of the PPT step.

In conclusion, the property of **Context** is addressed by POSE and the use of the pattern, and there is significant evidence in the thesis to demonstrate that POSE supports this property as indicated by Table 7.1.

7.1.2 The property Architecture

The property **Architecture** is the capability of an approach to structure the development model so as to resolve architectural issues. All of the reviewed SSSD approaches support this property and there is also support for it in the published literature [82].

The existence of SOLUTION INTERPRETATION and SOLUTION EXPANSION transformations highlight the importance of this property to POSE. This is consolidated in the Solution Interpretation and Expansion step (Activity 2 of the pattern) which allows candidate solution architectures to be evaluated – refer to section 4.3.

The Solution Interpretation and Expansion step was used in all of the case studies

to introduce candidate system architectures: the *DC* in section 4.3, the *SMS* in section 5.5.1, and the *FAS* in section 6.3.2. Further, this step was used again in the *DC* and *FAS* case studies to introduce modified architectures to solve the identified problems (*DC* in section 5.3 and the *FAS* in section 6.3.5).

In conclusion, the property **Architecture** is addressed in POSE and the use of the POSE safety pattern, and there is significant evidence in the thesis from the three case studies to demonstrate that POSE supports this property as indicated by Table 7.1.

7.1.3 The property Spec. Join

The property **Spec. Join** covers the ability of an approach to span both the problem space and the solution space. That is, the approach satisfies the property if it can develop the specification from the requirements in the problem space and then develop the solution from the specification in the solution space. This property is supported by all the SSSD approaches and in the published literature [121].

The POSE transformations CONTEXT INTERPRETATION and REQUIREMENT INTERPRETATION show the POSE capabilities in the problem space, whilst SOLUTION INTERPRETATION demonstrates some of the POSE capabilities in the solution space. The PPT work in section 5.1.2 showed that POSE can be used to derive a validated specification from requirements in context. This was demonstrated for a non-formal development in the *DC* case study of section 5.1.3 and section 5.4, and for formal developments in the *SMS* and *FAS* cases studies of sections 5.5.3–5.5.5 and section 6.3.3 respectively, and also in the rework of the *FAS* in sections 6.3.5–6.3.7.

As noted in section 2.6, the main focus of this work has been the early phases of the development process, hence it was classed as "ee" in Table 6.4. However, when the results of this work are considered together with the published POSE literature that shows how a POSE derived specification can be used to develop a solution (e.g.,

[41, 39]), we conclude that there is sufficient evidence to demonstrate that POSE supports this property. Therefore it is classed as "E" – significant evidence – in Table 7.1.

7.1.4 The property Avoid Bias

The property **Avoid Bias** relates to the avoidance of structures in the problem development that result in unnecessary and/or inappropriate constraints on the solution. The importance of this property is highlighted in the published literature [133, 16], and as detailed in section 2.5.2, avoiding solution bias is a characteristic of the PO approaches, which stress the importance of fully understanding the problem before designing a solution, and of expressing requirements solely in terms of the context phenomena.

The detailed presentation concerning REQUIREMENTS INTERPRETATION in section 4.1.2.2 and section 4.1.3.2, and introducing the solution in section 4.1.3.3 on the *DC* case study provide evidence to show that POSE supports this property. Further evidence is provided by the *SMS* case study in section 5.5.1 and the *FAS* case study discussion (section 6.4) which refers to section 6.3.1. The conclusion is that there is good evidence to show that POSE supports this property as indicated by Table 7.1.

7.1.5 The property Model Reality

The property **Model Reality** is the ability of the approach to represent the problem in terms of the phenomena, relations and concepts that exist in the real world. All the SSSD approaches support this property to some degree, it is supported in the literature [133, 16], and the concept of designating descriptions and phenomena in the real world is an important aspect of POSE (section 2.5.2).

For the *DC* case study the REQUIREMENTS INTERPRETATION work detailed in

section 4.1.2.2 and the context mapping discussed in section 4.1.3.1 provide evidence to support this property. Further evidence is provided by the *SMS* case study in section 5.5.1 and the *FAS* case study discussion (section 6.4) which refers to section 6.3.1. The conclusion is that there is good evidence to show that POSE supports this property.

7.1.6 The property Validation

The property **Validation** is concerned with the ability of the approach to ensure that the right system is being produced by applying analysis to the models as they are developed. All the SSSD approaches support this property, and some (e.g. SpecTRM, SCR, KAOS, AMBERS, PFS offer tool support for simulation and proof. There is also much support in the published literature including [122, 16]. Validation is an important aspect of POSE through its notion of transformation justification. The PPT work in section 5.1.2 shows how problem progression can be validated using the useful behaviour relation of the domain to be progressed. The *DC* case study demonstrated this for a non-formal development in section 5.1.3, whilst the *SMS* case study provided evidence for a formal validation in section 5.5.5. The *FAS* case study provides further support for validation using a formal development in section 6.3.4.2 and section 6.3.7. Further, one of the purposes of the POSE safety pattern introduced in section 6.1 was to allow an early effective safety analysis that determined whether the system model was capable of satisfying its safety requirements and would thus be a viable basis for the rest of the development. This safety validation was demonstrated on the *FAS* case study, and also in the PSA work reported for the *DC* and *SMS* case studies. The conclusion is that there is strong evidence to show that POSE (particularly in combination with Alloy) supports this property as indicated by Table 7.1.

7.1.7 The property Simplification

The property **Simplification** is the ability of the approach to manage complexity by structuring the problem, so that large complex problems can be broken down into smaller problems that are easier to solve. The SSSD approaches recognise the importance of this property, although only KAOS, AdLS and to a lesser extent AMBERS offer process support. There is also much published literature including [69, 16].

The *DC* case study in section 5.4 and the *SMS* case study in section 5.5.2 provide evidence that POSE supports this property. Problem Simplification is Activity 3 in the POSE safety pattern used in the *FAS* case study and this involves application of the PPT to make the problem of deriving a specification easier to solve (refer to section 6.3.3 and section 6.3.5). In conclusion, the case study results provide sufficient evidence to show that POSE supports the property **Simplification** as indicated by Table 7.1.

7.1.8 The property Tool Support

The property **Tool Support** relates to the efficiency, error-avoidance and repeatability advantages that can accrue if the approach has adequate tool support. Most of the SSSD approaches offer tool support and the desirability of tool support is noted in the published literature, e.g., [96, 45, 90]. As noted in the discussion in section 6.2.2, although POSE does not provide any tool support directly, this is mitigated by the ability of using POSE models within other tools, like Alloy. The *FAS* case study provided further evidence to support this view. In conclusion, although POSE would benefit from its own tool support, the ability of its models to be used within other tools means it is feasible to use it as part of the development process. This is an area that would benefit from further work, for example the development of the proof-of-concept tool mentioned in section 6.2.2 would be a useful improve-

ment – this will be discussed further in Chapter 8 – hence this property is "e" in Table 7.1.

7.1.9 The property Right Abstraction

The property **Right Abstraction** is the ability of the approach to operate at the level of detail appropriate to the problem at hand. For example the approach should be able to describe the range from high level requirements through to detailed specifications. The importance of this property is noted in the published literature, e.g., [82, 71].

Application of PSA to the POSE models in the case studies provide some evidence that POSE supports this property. Further support was provided by the *DC* case study in section 5.1.3 and the *SMS* case study in section 5.5.1. The discussion in section 6.4 indicates how support for this property is provided by the *FAS* case study. In conclusion, the case studies provide sufficient evidence that POSE does support this property as indicated by Table 7.1.

7.1.10 The property Integrates

The property **Integrates** is concerned with the ability of the approach to work and integrate with an existing development process. The *DC* case study work reported in section 4.5.2 indicated that the POSE transformations could be integrated with the IPDP, but that further supporting evidence was required. The discussion in section 5.6 noted that the *SMS* case study provided evidence to support the view that POSE can integrate well with IPDP. Further evidence of good integration is provided by the *FAS* case study discussion in section 6.4. The conclusion is that POSE integrates well with the IPDP as indicated by Table 7.1.

7.2 Discussion of the IPDP Issues and Properties

Chapter 3 described a successful process (IPDP) used to develop a variety of safety critical and safety related embedded real-time systems. Although IPDP has been successful, a number of issues with it and possible improvements to it have been identified through its use. These were identified in the chapter as six issues and four properties. The extent to which POSE can mitigate these issues and support these properties will now be discussed for each of them.

7.2.1 Mitigation for Issue [EnvirI1]

The first of the six IPDP issues, **[EnvirI1]**, concerns the risk that insufficient understanding of the system context increases the risk of later rework:

> **[EnvirI1]:** Incomplete understanding of the system context increases risk of rework.

In the past this has allowed ambiguity and lack of detail in the descriptions to result in misinterpretations of the design concept and further, an adequate safety analysis requires that the system environment is sufficiently understood. This issue is closely related to the property **Context** discussed in section 7.1.1, which details how understanding the environment context is a fundamental concept in POSE.

From an IPDP perspective the context understanding work corresponds to the CE phase. For example, in the *DC* case study the initial understanding work (section 4.1.2.1 and Appendix A.2) occurs at the beginning of the CE phase and this is refined and developed as the various CE phase tasks are completed leading to the use of the CONTEXT INTERPRETATION transformation to introduce context into the problem to form P_{Doms} (section 4.1.3.1). Towards the end of the CE phase work further understanding of the domains resulted in a series of additional CONTEXT

INTERPRETATION transformations that transformed the problem into P_{Dom} – these were validated by review. A similar process was followed in the *SMS* case study (section 5.5.1, discussed in section 5.6) and in the *FAS* case study (section 6.3.1 Activity 1 of the POSE safety pattern, discussed in section 6.4). In conclusion, POSE and the POSE safety pattern provide significant mitigation concerning this issue as indicated by Table 7.1.

7.2.2 Mitigation for Issue [RequI2]

The second of the six IPDP issues, **[RequI2]**, concerns the risk of rework associated with not adequately understanding and acquiring the requirements:

[RequI2]: Incomplete requirements increases risk of rework.

This increases the risk that the right system is not being produced and it has been observed following IPDP that the detailing of the requirements allocations and transformations do not always produce requirements that are feasible, nor are they necessarily sound, consistent or complete.

The issue **[RequI2]** is concerned with understanding the customer requirement. For example, in the *DC* case study the work begins in the IPDP CE phase (see section 4.1.2.2 and Appendix A.2) where the requirement is elicited and developed in conjunction with the customer. As system understanding increases the requirement is developed accordingly. In the case study a REQUIREMENT INTERPRETATION transformation was used to introduce the initial requirement, *RDC*, into the problem. This was developed into *Rdc* through a series of further REQUIREMENT INTERPRETATION transformations as detailed in section 4.1.3.2 as the tasks in the CE phase were progressed and the Dem/Val phase started (refer to Appendix A.3). The Initial Problem work in the *SMS* case study (section 5.5.1, discussed in section 5.6) and the *FAS* case study (section 6.3.1 Activity 1 of the POSE safety pattern, discussed in section 6.4) provide further supporting evidence. In conclusion, POSE

and the POSE safety pattern provide significant mitigation concerning this issue as indicated by Table 7.1.

7.2.3 Mitigation for Issue [LateI3]

The risk of rework caused by performing the safety analysis relatively late in the development life cycle was captured as the third IPDP issue:

[LateI3]: Late effective safety analysis increases risk of rework.

Developing the capability to perform an effective safety analysis earlier in the (IPDP) development process was one of the major drivers for this work. An effective analysis requires an accurate understanding of the context, the requirement and an appropriate model to analyse (see section 3.2.3, section 3.3.4 and section 2.2.2). The work in section 5.2 indicated how an improved PSA could be achieved for non-formal (section 5.2.1.1) and formal (section 5.2.1.2) developments using the POSE problem model available after problem progression (P_{Red}). The *DC* case study demonstrated the process for a non-formal development in section 5.2.2, whilst the *SMS* case study demonstrated the process for a formal development using POSE and Alloy in section 5.5.5. The *FAS* case study demonstrated how a formal PSA could be conducted as a result of following the POSE safety pattern in section 6.3.4 and in sections 6.3.6–6.3.7. The work demonstrated that an effective safety analysis can be achieved as long as P_{Red} is available.

In the original *DC* development (Appendix A.3) FFA was applied during the Dem/Val phase, but this missed an issue – HAZOPS was not applied until Preliminary Design in the EMD phase. In section 4.4 PSA was applied using FFA and FTA, whilst in section 5.2.2 a combination of FFA, HAZOPS and FTA were used. The latter required that the context (domains and phenomena) and requirement information was available to support a HAZOPS. The information collected to support the development from P_{null} through to P_{Red} ensured that the required information

was available. The work in section 6.1 noted that by structuring the IPDP tasks through the use of the POSE safety pattern then it could be ensured that the necessary information to support a comprehensive PSA (i.e., one using the techniques introduced in section 5.2) could be made available at the end of the Dem/Val phase, prior to the SDR. This work demonstrates that an effective safety analysis can be applied at an earlier part of the IPDP (end of Dem/Val, prior to SDR) than was achieved with the existing IPDP (after SDR).

The results of the revised *DC* case study in Appendix H provides further evidence to corroborate that this property has been satisfied – this additional evidence progresses the "ee" in Table 6.4 to "E" in Table 7.1. The conclusion is that POSE and the POSE safety pattern provide significant evidence to mitigate this issue.

7.2.4 Mitigation for Issue [TraceI4]

The fourth IPDP issue, [**TraceI4**], concerns traceability incompleteness and can be split into two main themes: traceability of requirements through the design, and the ability to trace back through a design to establish what information can be re-used and what needs to be developed, including the ability to know where to trace back to.

> [**TraceI4**]: Traceability incompleteness increases risk of rework.

The ability to support iteration is also considered a form of traceability issue by this work.

The work in Section 5.3 introduced an enhanced problem transformation trace graph as a graphical representation of the relationship between the software problem descriptions and their associated requirements transformations – an example is shown in Figure 5.4. This graphical form allows the path of the requirements transformations to be traced from the original high level system requirements through to

the detailed requirements that directly support the specification derivation, based on the software problem transformation sequence.

The work on the *DC* case study in section 5.3 provides strong evidence that both aspects of the issue **[TraceI4]** are mitigated by the POSE transformations and use of the pattern. The graph was also used in the *SMS* case study (section 5.5.2) to represent the development and simplification (problem progression) sequence from $P_{Initial}$ to P_{Red}. The graph in Figure 6.7 performed a similar role on the *FAS* case study, except it was also used to assist with the backtracking (see section 6.3.5). In conclusion, the POSE and POSE safety pattern provides good evidence to mitigate this issue although more development work of these traceability techniques is required to realise their full potential – this will be discussed further in Chapter 8 – hence this issue is "ee" in Table 7.1.

7.2.5 Mitigation for Issue [IncomI5]

The fifth IPDP issue concerns the problems associated with not completing all of the work items associated with a task and was captured as:

> **[IncomI5]:** Task incompleteness increases risk of rework.

Task incompleteness includes the problems associated with tailoring out (removing) tasks from the schedule.

The *FAS* case study in section 6.3 demonstrated that the POSE safety pattern organised the development tasks such that a successful PSA could be applied before the end of the Dem/Val phase. The pattern activities ensure that, at a high level, all the necessary tasks to support the formation and feasibility/validation checking of a problem model are undertaken and in the correct sequence. Further, each activity is associated with a problem transformation step which ensures that all the necessary tasks associated with that activity are completed. For example, PS1 on page 91 includes the justifications for the transformation sequence ($J_1 = J_a \wedge J_b \wedge J_c$), and the

necessary concerns/claims/evidence that the interpretations are well-founded, the system architecture is feasible, sound judgement followed and important equipment has the necessary reliability.

Inspection of the problem transformation steps shows each is associated with a set of concern/claim/evidence elements that validate a particular concern associated with the use of that step, and this mechanism could be developed further to ensure that the necessary tasks to support a transformation step are included in the structure used for that step. In conclusion this thesis work has demonstrated that the POSE safety pattern and the problem transformation steps it uses provide some evidence to mitigate this issue, but that further opportunities exist to improve this mitigation further - these will be discussed in more detail in Chapter 8 – hence this issue is "e" in Table 7.1.

7.2.6 Mitigation for Issue [ValidI6]

Validation is a key part of the development process and is captured as the sixth IPDP issue as:

> **[ValidI6]**: Validation incompleteness increases the risk of rework.

Validation includes validating that the requirement (as it is interpreted through the development) meets the customer's expectations, i.e., that the right system is being developed.

The PPT for safety systems introduced in section 5.1.2 and the *SMS* case study provide mitigation for this issue. Further, the Validation property considered in section 7.1.6 shows how POSE supports validation in a general sense, but validation is also an important feature of IPDP and features prominently in all the phases. The POSE safety pattern organises the POSE problem transformation steps so they integrate with their associated IPDP tasks to ensure that an effective PSA and a specification can be derived prior to SDR. This is discussed in section 6.1 and

demonstrated in the *FAS* case study in section 6.3 - particularly the validation work in section 6.3.4.2 and section 6.3.7. The work in section 6.3.7 demonstrates that the problem model satisfies its requirements, satisfies its safety requirements and the requirements are shown to be consistent. In addition, the revised *DC* case study in Appendix H provides further evidence to mitigate this issue. This additional evidence progresses the "ee" in Table 6.4 to "E" in Table 7.1. In conclusion, this thesis work has shown that POSE and the POSE safety pattern provide significant mitigation for this issue.

7.2.7 Support for [NecessaryP]

The necessary property identified in Chapter 3 , **[NecessaryP]**, requires that:

> **[NecessaryP]**: The approach used must be capable of deriving design specification.

which is a necessary feature of any approach to be used in the early phases of embedded system development. The POSE PPT developed in section 5.1.2 demonstrated how problem progression could be applied to remove domains to simplify a problem under certain conditions: that causal relations among phenomena are identified and used to establish assumptions in the derived requirements; that only domains whose phenomena are adjacent to the requirement are the subject of progression. The first criterion is reasonable for the real-time embedded avionics systems that are the subject of this work, and the second follows from the application of the POSE safety pattern which ensures progression (Activity 3) follows after Activity 2. The output of Activity 2 is a model (called P_{Exp} in the case studies) in which the requirement is well understood, interpreted, validated and at the right level of abstraction, thus forming an appropriate candidate requirement for the initial progression. The PPT then removes the domains in turn, each time re-interpreting the requirement into an appropriate form and capturing the useful behaviour relation of the progressed

domain as an assumption. This allows the new requirement to be validated against the original requirement through the assumption.

The work in section 5.1.3 demonstrated the use of the PPT to simplify a non-formal development problem, where the validation was completed using an informal review process. In the *SMS* case study of section 5.5 the PPT was applied in a formal development, and the final step was formally evaluated (i.e., that the derived specification satisfied the requirement of the problem model P_{Red}) in section 5.5.5. A formal approach based on simulation and proof was used in the *FAS* case study to validate that the specification derived from the repeated application of the PPT (Activity 3 of the POSE safety pattern) satisfied its requirement – refer to sections 6.3.3–6.3.4 and also sections 6.3.5–6.3.7 for the rework. In addition, the revised *DC* case study in Appendix H provides further evidence to support this property. This additional evidence progresses the "ee" in Table 6.4 to "E" in Table 7.1. In conclusion this thesis work has provided significant evidence to demonstrate that POSE and the POSE safety pattern support this property.

7.2.8 Support for [SameA1]

The desirable property that promotes efficient analysis identified in Chapter 3, **[SameA1]**, requires that:

> **[SameA1]:** The safety analysis uses the same model as the rest of the development.

In the *DC* case study the PSA was applied to the P_{Exp} model in section 4.4 and to the P_{Red} model in section 5.2.2. In both cases the PSA model was the same as the development model. The *SMS* case study in section 5.5.5 and the *FAS* case study in section 6.3.4 and sections 6.3.6–6.3.7 provide further evidence to support this property. In conclusion this thesis work has provided significant evidence to demonstrate that POSE supports this property as indicated by Table 7.1.

7.2.9 Support for [FormalA2]

The desirable property that promotes effective analysis identified in Chapter 3, **[FormalA2]**, requires that:

> **[FormalA2]:** The approach used should support formal development processes.

The *SMS* case study in section 5.5 and the *FAS* case study in section 6.3 both provided strong evidence that POSE works well with the Alloy formal system, such that POSE problem models could be directly encoded into Alloy to allow formal validation and proof to be applied. That is, the thesis work showed a good synergy between the POSE problem model and the Alloy formal system, and this included the following features of note:

1. an Alloy model formed directly from the structure of the POSE problem model (using domain and phenomena knowledge).

2. a timing model for use with the Alloy simulation was developed to allow fine control over the setting of phenomena values.

3. a four part analysis approach was developed based on:

   - the use of simulation to develop the model and gain confidence it has the intended operational behaviour;
   - the development of formal versions of the requirements and proof that the model satisfies them – the latter also validates consistency;
   - the development of safety properties (if required) and proof that the model satisfies them;
   - the use of simulation to explore problem situations, e.g., rapidly changing inputs.

In addition, the revised *DC* case study in Appendix H provides further evidence to support this property. In conclusion this thesis work has provided significant evidence to demonstrate that POSE in conjunction with Alloy supports this property. However, more work is required to evaluate how POSE operates with other formal techniques and this will be discussed further in Chapter 8 – hence this property remains "ee" in Table 7.1.

7.2.10 Support for [IntegA3]

The desirable property that promotes an efficient development identified in Chapter 3, **[SameA1]**, requires that:

[IntegA3]: The approach used should integrate with IPDP.

This property is directly related to the property **Integrates** covered in section 7.1.10, and the analysis presented there applies here. From an IPDP perspective, Table 4.3 provides strong evidence that POSE integrates well with IPDP. This is confirmed in the Chapter 6 work on the POSE safety pattern and how it integrates with the IPDP (see section 6.1), and in the *FAS* case study which shows the successful application of the pattern. Finally, further evidence to support this property is provided by the industrial case study discussed in more detail below. In conclusion this thesis work has provided significant evidence to demonstrate that POSE supports this property as indicated by Table 7.1.

7.3 Industrial Case Studies

The development of the POSE safety pattern, summarised in section 6.1, and the successful use of it on the case study in section 6.3 provided enough confidence to use it in the form of the POSE/Alloy combination at the front end of a number of real development tasks. The first two examples were modules from the same

safety related LRU, the third was a full safety critical LRU system development of a stores management system (SMS). Further, the fact that this actual SMS was similar in structure to the SMS case study investigated in section 5.5 provided additional confidence for using the pattern. The goal in applying the pattern was to provide additional complementary evidence to support the normal IPDP activities and in particular to provide early evidence that the proposed design was capable of satisfying the safety properties identified for it. The pattern was applied during the IPDP Dem/Val phase and the safety analysis work was completed prior to SDR - thus satisfying the early safety analysis issue represented by **[LateI3]**.

The first trial of the POSE safety pattern involved modelling two safety related modules of a communications control and routing function in a fast jet aircraft - these being the control module and the warning tones module (WTM). The POSE/Alloy modelling of the control module identified some requirements issues which were subsequently resolved. The modelling of the WTM identified a shortfall in the safety analysis task which was addressed. The WTM consisted of 41 requirements (excluding BIT), of which 27 of these were functional requirements. The POSE safety pattern was applied to these functional requirements and this produced an initial POSE safety problem description to which problem simplification was applied to allow the specification of the WTM to be derived directly. The latter was used to form an Alloy model which had a total of nine phenomena and the four-part process described in section 6.4 was applied. The POSE modelling of the WTM from first read of the requirements document through to being ready to begin the Alloy modelling process took one day of effort. Setting up and validating the Alloy simulation model took a further day of effort and this was followed by just under two days of effort to perform the proof and safety analysis tasks – about four days in all. The control module had a few more requirements, more phenomena (14) and a more complicated behaviour, but this was still fully analysed in just over a week (five days) of effort. The usual IPDP effort for this work was one to two days

to perform an in-depth manual review of the document, but this standard IPDP approach did not identify the issues.

The third trial of the POSE safety pattern "in the real" was the full modelling of a SMS development - this was a much larger enterprise being based on a 26 page Z model and involving 30 phenomena. Due to the complexity of the behaviour, the Alloy modelling was partitioned into a number of major functional areas to avoid memory resource problems with the simulation work. The modelling work identified nine issues with the requirements and their interpretation, which were resolved before development progressed. The POSE/Alloy modelling effort took two weeks and this was run in conjunction with the normal (IPDP) development process. The normal process effort for this phase would have been two to three days - but this did not identify the nine anomalies!

One of the complications in performing this work was overcoming the limitations imposed by the Alloy tool. The Alloy modelling was applied by successively adding functionality until the full behaviour was covered, but this increased complexity often resulted in the tool running out of memory resources as the number of tokens in the model increased. Strategies for addressing this problem included abstraction (e.g. combining phenomena with similar functionality to reduce the number of inputs) and slicing - where the overall functionality was broken down into its major functions, and these major function "slices" were modelled individually. As noted above, the latter was used on the SMS task.

Unfortunately due to security and confidentiality reasons little of the detail concerning these industrial case studies can be recorded here, apart from the fact that in all three cases the application of the pattern prompted a series of questions concerning the interpretation and clarification of: the requirements, the proposed system architecture and the safety analysis. The answer to these questions improved the early modelling work, the key point being that to produce the simulation model required that much of the ambiguity in the textual requirements had to be resolved.

This improved the quality of the requirements and the integrity of the design based upon them, by ensuring (and validating) that the requirements modelled reality and avoided any implementation bias – thus addressing the properties Model Reality, Validation and Avoid Bias from Table 2.3.

The industrial case studies also provided further strong evidence that POSE, through the use of the pattern, integrates well with the IPDP – thus satisfying the properties **[IntegA3]** (see section 3.4.5) and **Integrates** (from Table 2.3). The use of the pattern on the industrial case studies was carefully monitored to identify any task incompleteness issues. No such issues were identified and the conclusion was that the pattern, and the problem transformation step structure used in the pattern were very good at ensuring that all the necessary tasks were identified. Additional review items were added to the IPDP SDR review checklist to ensure that these identified tasks were successfully discharged. This work provided further direct supporting evidence for mitigating the issue **[IncompI5]**.

Finally, this section has discussed the use of the pattern on actual industrial case study developments – where POSE was used in parallel to the actual development, but did provide safety analysis and evidence to support the early phases of the development. This demonstrated that POSE could be scaled up to support real development activities and that it did complement and improve the existing IPDP.

7.4 Development Process Context

Many developments have a target technology or group of technologies as an overall approach since using well understood technologies can significantly reduce the overall project risks and is consistent with the normal design ideas discussed in section 1.1. This is the case at the author's Company, where the goal with the avionics developments is to use a hardware architecture consisting of processor and input/output card assemblies as the target technologies. The design function then

revolves around modifying the functionality to achieve the requirements, with the benefit that the underlying card assembly technology is well understood and known to be capable of success – significantly reducing unknowns and de-risking many aspects of the project. Therefore such projects include elements of bottom-up as well as top-down in the development process.

One of the advantages of using the POSE safety pattern is that it fits well with this approach of targeting known hardware architecture technologies through its use of the target machine specification, which defines the specification for the generic computing element which maps to the hardware processor card in the design. The POSE safety pattern aims to capture essential elements of the requirements and context, define the machine specification and validate that the high level system architecture represented by the POSE model is capable of satisfying its safety obligations. However it is used within the context of a top-down and bottom-up design approach. Further, although the derivation of the machine specification, S, from the requirement, R, using the entailment sequent $W, S \vdash R$ is top-down, the validation from S to R is bottom-up. Basically the POSE safety pattern application is the first phase of a top-down/bottom-up process that seeks to map the users requirements into a well understood target hardware architecture technology.

Further, the safety analysis associated with the application of the POSE safety pattern is essentially a de-risking exercise to validate that it is feasible to and hence worth continuing with the design process. As the development progresses more design decisions and target technology will be included into the design which will require further safety analysis to validate that it remains capable of satisfying its safety obligations.

7.5 Summary of Research Contribution

The aim of this section is to consolidate and summarise my research contribution on using POSE in the early stages of a safety critical system development as detailed in this document. My research covers the three areas of: theory development, developing the means of application, and providing initial or partial validation of the approach. In more detail, my contribution has three main parts:

- developing and extending the theory of Problem Orientation;
- applying the extended theory in a safety critical engineering setting;
- providing partial validation for the extended theory within that setting.

Together these discharge my overall research aim (refer to section 1.2), which can be summarised as:

> "Improving the safety process by providing the means of applying safety analysis to the early stages of a development."

The validation is only initial or partial as more extensive case studies and trials are required before full validation of the approach is feasible. However, I would argue that the partial validation achieved is commensurate with the work associated with the thesis development. The latter consisting of the case studies and some industrial use (as reported in section 7.3) – hence it is termed part-validation in the remainder of this section and in Appendix I which provides further details on my contribution.

7.5.1 Develop and extend the Theory

POSE, as it was, was of mainly theoretical interest lacking evidence of its applicability and use on realistic examples that might validate its practical use. Firstly, it lacked any guidance as to which transformation to apply next. This made it unsuitable for real engineering. My first contribution was to extend the theory with

knowledge of how to sequence the transformations applied. This was the topic that was first published in FASE07 [87] and developed in subsequent papers.

Secondly, at that time, there was a theoretical presentation of problem progression along the lines suggested by Professor Jackson, but from an engineering perspective the details were sketchy. My second contribution was to provide detailed guidance as to how to use problem progressions for safety critical engineering – published in the FASE07 [87] paper for informal presentation, and developed in detail in the IWAAPF08 [83] paper.

7.5.2 Demonstrate the applicability of the extended Theory

When I started, other than for some toy examples, POSE was not extensively tested through application. My third contribution to knowledge is given by the development of case studies to test POSE's applicability and the applicability of the extensions to POSE that I provided (of course, the process was iterative, and theory extension and application went hand in hand).

There were three main case studies; all realistic, taken from practice and used in the published papers. All were retrospective but demonstrated that POSE could be applied in an engineering context and could find the issues of interest – as discussed in this thesis for the SMS case study.

- FASE07 [87] – *DC* Case Study in Chapter 4 and Chapter 5.
- HASE07 [35] – *DC* Case Study in Chapter 4 and Chapter 5.
- SAFECOMP07 [86] –*SMS* Case Study in Chapter 5.
- IET07 [85] – *FAS* Case Study in Chapter 6.
- IWAAPF08 [83] – *FAS* Case Study in Chapter 6.
- SSS08 [88] –*FAS* Case Study in Chapter 6.

My fourth contribution to knowledge concerned applicability with respect to the Research Aim and included the development of safety analysis technique improvements and the derivation of formal models based on the capabilities provided by extended POSE. This work also included developments to support the formal modelling work.

7.5.3 Partial Validation of the extended theory in an engineering setting

My fifth contribution to knowledge is the improvement of current engineering practice. We have shown how the engineering extended POSE forms a good fit for IPDP, and we have assessed it as improving aspects of IPDP. As IPDP is typical of processes within the industry we consider that there is some evidence that such support can assist processes based on similar safety standards and regulatory frameworks.

In a nutshell, the work has taken a promising theory and extended it to produce an approach suitable for improving the early phases of a safety development process.

7.6 Conclusion

The aim of this work was to demonstrate that POSE, with the extensions introduced in this work, could satisfy two main objectives. The first was that it could address the ten SSSD approach properties identified in section 2.5 and thus be used as an SSSD approach that is appropriate for safety system development. The second objective was that it could be used to improve the early phases of a typical safety development process by mitigating or overcoming issues identified with the requirements engineering aspects of the IPDP. The required enhancements to the IPDP were identified in section 3.4.5 in terms of six issues to be mitigated and four properties to be supported. The discussion points in section 7.1 provide sufficient

evidence to show that POSE and the POSE safety pattern address the ten SSSD approach properties and thus satisfy the first objective. The discussion points in section 7.2, with further supporting evidence from section 7.3, demonstrate that POSE and the POSE safety pattern adequately mitigate the issues and support the properties such that they do improve the early phases of the IPDP, and thus satisfy the second objective. The case study work also demonstrated that POSE could be used for non-formal and formal safety system developments.

The work reported in [32] (see section 2.3.3) to provide enhancements to the Parnas 4-Variable model are compatible with the POSE approach presented in this document. In particular the "outside" enhancements described in [32] map well to the development of the POSE model, which also distinguishes the embedded system represented by the specification S, the environment context represented by W and the platform represented by the requirement R through the sequent $W, S \vdash R$ and handles the indirection issue through the non-feasible requirements mechanism.

Inspection of the case study work shows that the POSE and Alloy combination were used successfully in the early phases of the development life cycle, which is consistent with when the "lightweight" PSSA identified in section 2.3.8 should be applied. Therefore, the work presented here could be integrated as the front-end of the FLM/FI approach outlined in [77].

The contribution that this work makes is summarised at the end of each of Chapter 4 (Table 4.5), Chapter 5 (Table 5.6) and Chapter 6 (Table 6.5), and is presented in more detail in section 7.5 and Appendix I. The contribution has three main parts: developing and extending the theory of Problem Orientation; applying the extended theory in a safety critical engineering setting; providing partial validation for the extended theory within that setting. The summary of the work presented in section 7.5, with further detail in Appendix I, is that POSE, as extended by the research work reported in this thesis, does satisfy the research objectives as listed in section 1.2 and thus satisfies the overall goal of:

"Improving the safety process by providing the means of applying safety analysis to the early stages of a development."

Property/Issue	Previous Work	Chapter 4	Chapter 5	Chapter 6	Chapter 7
Context	[41, 87]	e	ee	E	**E**
Architecture	[41, 87]	e	ee	E	**E**
Spec. Join	[39, 83]	-	e	ee	**E**
Avoid Bias	[39], [Ind_Case]	e	ee	E	**E**
Model Reality	[Ind_Case]	e	ee	E	**E**
Validation	[Ind_Case]	e	ee	E	**E**
Simplification	[41, 87]	e	ee	E	**E**
Tool Support	[35, 87]	-	-	e	**e**
Right Abstraction	[41, 87]	e	ee	E	**E**
Integrates	[88], [Ind_Case]	e	ee	E	**E**
NecessaryP	[41, 83]	-	e	ee	**E**
SameA1	[88]	e	ee	E	**E**
FormalA2	[41, 88]	-	e	ee	**ee**
IntegA3	[Ind_Case]	e	ee	E	**E**
EnvirI1	[87, 88]	e	ee	E	**E**
RequI2	[87, 88]	e	ee	E	**E**
LateI3	[85, 88]	-	e	ee	**E**
TraceI4	[85, 88]	-	e	ee	**ee**
IncomI5	[Ind_Case]	-	-	e	**e**
ValidI6	[88]	-	e	ee	**E**

Table 7.1: Final Evidence Status for POSE

Table 7.1 provides the final evidence status for POSE, it extends Table 6.4 by including additional evidence from the industrial case studies reported in section 7.3 (shown as [Ind_Case] under the "Previous Work" column), and the revised *DC* case study in Appendix H – where the entries for **Simplification**, **NecessaryP** and **LateI3** are further corroborated, as described in section 7.1 and section 7.2. This additional support is incorporated into Table 7.1.

In conclusion, this work has demonstrated that POSE and the POSE safety pattern can be used in the early phases of a safety system development to produce a validated specification that can be subjected to a comprehensive safety analysis to show that it satisfies its identified safety properties, and thus forms a viable basis

for the rest of the development process.

Chapter 8

Future Work

The conclusion presented in Chapter 7 was that this work has demonstrated that POSE and the POSE safety pattern can be successfully used as an SSSD approach for developing safety systems and can also be used in conjunction with an existing safety development process to achieve improvements in the early development phases. The research work has identified a number of areas that would benefit from further investigation and these will now be discussed.

The first area concerns the direct development of the Alloy formal model from the POSE requirements model that is formed by following the pattern. Alloy was selected for this work because it had the required capabilities, is easily available and has an active user community. However it is not the only candidate; as noted earlier in this work, the B-tool [111] has many positive characteristics, not the least being that it is being used in industry to address real safety problems. Further Z [116] is still being used actively for safety development work. Z has the advantage that it relates well to the Praxis SPARK toolset [7] (used with the IPDP) and is also well known to the high integrity development software engineers who use the IPDP. Therefore an important area of future work is to investigate the use of the pattern with other formal techniques used by the safety community to establish if a similar level of synergy can be achieved as that obtained from the POSE/Alloy

combination. A particularly important aspect is whether it is as easy to develop the formal models from the POSE requirements model.

Related to this is the important area of tool support. A prototype POSE tool has been developed to support the POSE transformations and facilitate the development of a POSE requirements model. This could be further developed in a number of important areas to provide: facilities to support the correct use of the pattern could be built into the tool – including the step templates; checks to ensure that the necessary validation text has been entered for each of the problem transformation steps[1] could be included; facilities to produce Alloy code templates to facilitate the derivation of the Alloy model from the POSE requirements model could be developed. The output of the latter would be a text file containing the phenomena and predicate names – the analyst would then add the behavioural information.

Experience of using the POSE safety pattern when feasibility issues are identified, indicates that the location of where to iterate back to follows easily from inspection of the problem transformation graph and that much of the original work can be re-used. This notion was explored in the Warning System *FAS* case study in Chapter 6, where it was shown that even in the problem transformation steps that needed to be updated (WPS2 and PrePSA) the changes were quite localised and most of the rest of the activities in the step could be read across from the original. The remaining steps could be re-used directly with only the additional phenomena and derived requirement needing to be added. Further work is required to demonstrate that these benefits are general, however the initial indications from this case study and other work are that this is a fruitful area for further research. Therefore in summary, when re-work is required the POSE safety pattern structure allows much of the original design and validation work to be re-used so that only the modified areas require re-work and the problem transformation step structure identifies these

[1]This is likely only to be a syntactic check that text has been entered; but a useful check nevertheless.

areas clearly. This allows change to be enacted with minimal impact, making this iteration efficient.

The discussion in section 7.2.5 noted how the POSE safety pattern and the problem transformation steps it uses provide a structure that can be used to ensure that all the necessary tasks are started and in the correct sequence. That discussion also noted that the concern/claim/evidence mechanism can be used at a more detailed level to ensure that the necessary tasks to support a transformation step are included in the structure used for that step. Moreover, it was noted that each type of problem transformation step (e.g., an Initial Problem step or a PPT step) was associated with certain concerns and that the set of concerns was different for different step types. For example, the PS1, SPS1 and WPS1 are all Initial Problem transformation steps and they have similar concerns. In contrast, PS1 and PS2 are different types of step and they have different concerns. This information could be used to produce problem transformation step templates for each of the steps used in the POSE safety pattern – namely the Initial Problem, Solution Interpretation and Expansion, PPT and PrePSA (used at end of Activity 3) steps. The concerns for each type of step would be included in its template, thus providing a further mechanism for ensuring that all the necessary tasks are completed. Therefore exploring the development of problem transformation step templates would be a useful area for future work.

The safety standards (refer to section 2.2) require a safety case to present the safety argument that the system being developed is safe for entry into service. In Europe, and increasingly further afield [34], GSN is the preferred way for presenting this safety argument (refer to section 2.2.1). Further, the standards expect the safety and development life-cycles to be appropriately integrated (e.g., [125]). The Assurance Based Development (ABD) approach [117, 34] is a means of achieving this integration through the context of the safety argument, and the expectation is that the full benefits of using the POSE safety pattern will be achieved by following

ABD or a similar approach.

A major advantage of integrating the safety case generation with the system development is that issues can be identified and mitigated as they arise [34], minimising the impact on the development schedule. Finding problems later in the life cycle is always more expensive and problematic. Further, this integrated approach is both more efficient and effective [34], since it avoids the waste of producing evidence that does not support an argument (hence it is not needed), and in conjunction with the use of the concept of SALs (Safety Assurance Levels) [132], it provides sufficient evidence to support the safety argumentation.

Determining system context accurately and completely is a major element of ABD [117], and this aspect conforms to Activity 1: Initial Problem Understanding of the POSE safety pattern. Evaluating candidate architectures is also an important part of ABD [34], which identifies seven criteria for determining the suitability of an architecture - including feasibility. This area is covered by Activity 2: Solution Interpretation and Expansion of the pattern, with the feasibility being checked by the pattern PSA. Further, ABD and the POSE safety pattern are complimentary in that they both allow the major effort to be focussed on the key safety areas. One of the problems with applying formal techniques is there is little incentive to use them if their benefits (in supporting the safety argument) are not clear [34]. However, as noted earlier, the Alloy formal model follows directly from the POSE problem model and has the defined goal of providing evidence to support the feasibility of this POSE problem model. Therefore in this context formal methods are efficient to use and effective, because their benefit is to provide direct evidence with respect to the feasibility of a candidate design.

A further advantage of ABD is that it is known to work well with problem oriented approaches, the successful synthesis of an assurance case with Problem Frames being reported in [117], noting that the techniques were complementary as each enhances the other by addressing weaknesses and shortfalls. Further, their

synthesis is accomplished with the artefacts that must be created in the normal course of system development, i.e., no new additional artefacts are required. Hence the process is efficient as well as effective. Initial work with POSE indicates that using the ABD approach with the safety argument in GSN and the pattern used to drive the development could provide an effective, integrated and efficient safety process and would benefit from further research and development.

Appendix A

IPDP applied to the *DC* Case Study

A.1 Problem Description and Scope

The goal of the first case study is to demonstrate the tasks and activities involved in applying the Company's IPDP to a typical safety critical development. This case study is based on the development of a Decoy Controller (*DC*), which is part of the defensive aids suite (DAS) on an aircraft. Only the initial phases of the IPDP up to the end of PDR (refer to Figure 3.1) are covered as these phases form the focus for this work. As the case study was developed following IPDP, so the first set of activities were concerned with the IPDP CE phase.

UK military applications make use of DS 00-56, but a similar hierarchy of safety analyses takes place for other safety standards and guidelines (e.g. ARP4761 [109]). For aircraft applications typically the Company provides a Line Replaceable Unit (LRU). The LRU is a system component that slots into one of the major sub-systems on the aircraft. For example, the Decoy Controller (*DC*) LRU introduced in section 4.1.1 below is part of the Defensive Aids Suite (DAS) sub-system on the aircraft. The aircraft manufacturer (and the Company's customer) is responsible for

the aircraft level safety analysis (equivalent to the Aircraft FHA in ARP4761) and for the major sub-system safety analysis (equivalent to the System FHA). These FHA analyses identify and allocate the safety targets that the individual LRUs that form the sub-system must satisfy. For example, for the DAS sub-system, the customer identifies the safety targets **H1** and **H2** that the *DC* LRU must satisfy. These targets are flowed down to the Company as the safety requirement that the *DC* must comply with. Thus the early safety analysis tasks undertaken in IPDP that form the major focus of this work, correspond to the PSSA phase of ARP4761.

A.2 CE Phase for the DC Case Study

The first activity in CE involves gaining an understanding of the system context, which includes understanding the system environment, identifying what the main components of the system environment are, what they do and how they interact with each other. The requirement of the proposed system is then elicited and the safety requirement analysed with respect to the system context. All this information is then collated, which allows a candidate conceptual design to be formed. The conceptual design for the *DC* consists of the block diagram shown in Figure A.1 together with the ensuing descriptive text, informed by the requirement (also presented below).

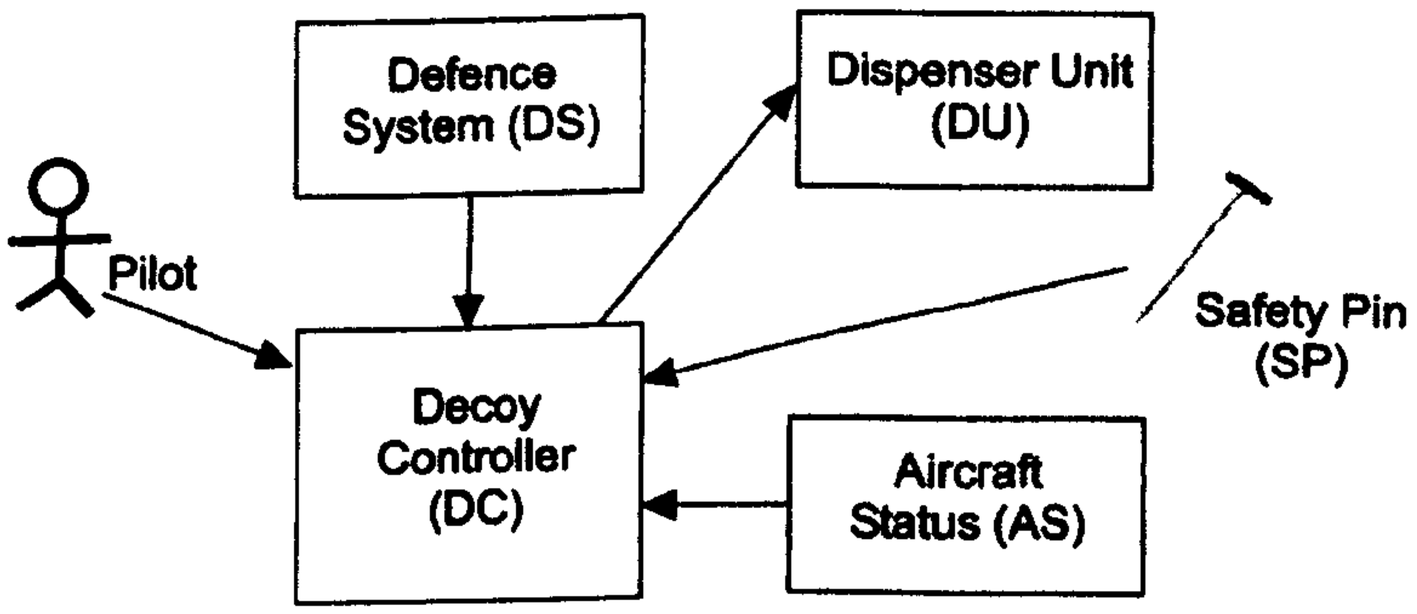

Figure A.1: Decoy Controller (DC) Block Diagram

The role of the *Decoy Controller* (*DC*) is to control the release of decoy flares providing defence against incoming missile attack. The *DC* interfaces with the *De-*

fence System (*DS*) computer, which is responsible for controlling and orchestrating all the defensive aids on the aircraft. The *DS* and other domains (see Figure A.1) already exist, and the task is to design the *DC*.

The conceptual design was discussed with the Customer and the initial functional requirement was revised and agreed accordingly.

The Customer FHA work (at the aircraft and major sub-system level) identified two safety hazards which were allocated to the *DC*, concerned with the inadvertent firing of the decoy flares, as follows:

H1 Inadvertent firing of decoy flare on ground. Safety Target: safety critical, 10^{-7} fpfh[1]; and

H2 Inadvertent firing of decoy flare in air. Safety Target: safety critical, 10^{-7} fpfh.

Further preliminary safety analysis work was conducted using the conceptual model and the hazards **H1** and **H2** as a basis. The model was explored to ascertain if any other hazards or safety issues might be relevant. This involved using the domain knowledge of experienced engineers, comparison with similar systems and a brainstorming session which was used to bring this knowledge and experience together. The conclusion of this work was that **H1** and **H2** were the pertinent hazards for this system as represented by the conceptual model. It should be noted that the conceptual model is high level and abstract. It is sufficient for generating a hazard list, but not detailed enough to discharge the obligations raised by the remaining IPDP preliminary hazard analysis tasks. These require the more detailed models generated by the later phases of the IPDP. This means that there is always a risk that further safety issues may be identified later in the life cycle, with all the attendant cost and schedule impact that this might have. Theses hazards were then collected together to form the *DC* system safety requirement **RS**. As a result of this work the customer requirement *RDC* for the *DC* can be expressed as follows:

[1] *fpfh* is 'failures per flight hour.'

RA The *DS* shall command which flare is selected for release by using a field in the *con* message it transmits.

RB The *DS* shall command the *DC* to issue a fire command in its *con* message. This shall be the only way in which a flare can be released.

RC The *DC* shall cause a flare to be released by issuing a fire command to the *DU*, which will fire the selected flare.

RD The *DC* shall only issue a fire command if its interlocks are satisfied, i.e. aircraft is in air, the *SP* safety pin has been removed and the *Pilot* has issued an allow a release command.

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

Therefore, the overall requirement is $RDC = RA \wedge RB \wedge RC \wedge RD \wedge RS$

A complete statement of **R** should also include requirements that cover space, weight, environmental performance, interfaces and so on, but these are beyond the scope of this work. The SRR validated the high level system and safety requirement detailed above as being suitable for the next phase of the development – Dem/Val.

A.3 Dem/Val Phase for the DC Case Study

The initial part of the Dem/Val phase involves analysis and modelling to gain a better understanding of the system and its environment. This includes more detailed analysis of the requirement and an analysis of the safety requirement. A summary of this work for the *DC* case study now follows. The hazards **H1** and **H2** have both systematic (safety related) and probabilistic components. The initial safety analysis work identified that to counter these hazards the architectural design of the defensive aids system should contain a number of safety interlocks in the *DC* to provide safety protection. These are: an input from the pilot indicating whether

the release should be allowed; an input indicating whether the aircraft is in the air; and an input indicating whether the safety pin, present when the aircraft is on the ground, is in place. The expected behaviour is that flare dispensing should be inhibited if any of the following conditions hold: a) the pilot disallows flares; b) the aircraft is not in the air; or c) the safety pin has not been removed. It transpires that the interlocks b) and c) provide extra assurance for hazard **H1** but not for **H2**. Therefore, the safety task is to demonstrate that **H2** can be satisfied using just interlock a), with the knowledge that if **H2** can be satisfied, then so can **H1**. Note, flare selection and timing are not safety related, it is only applying an inadvertent fire command to any flare that is regarded as a safety issue.

The analysis and modelling work, reported in the previous paragraph, is collated to produce a system theory of operation document that is also descriptive text. This is validated with the customer before embarking on the next set of tasks which involve more detailed architectural analysis and the development of the sub-system theory of operation. The functional part of the sub-system theory of operation task is presented below for the *DC* system.

Sys The Decoy Controller (*DC*) system consists of the *DC*, the Pilot selection switch (*Pilot*), the Dispenser Unit (*DU*), the Aircraft Status (*AS*), the Safety Pin status (*SPS*) and the Defence System (*DS*) as shown in Figure A.1.

Pilot The *DC* receives an input from the Pilot, which is a boolean switch selection. When the pilot selects the switch to on (Pilot = on) this means that the dispensing of flares is allowed by the *DU* under the control of the *DC*. If the pilot selects the switch to off (Pilot = off) then the *DC* will inhibit the dispensing of flares by the *DU*.

AS The *DC* receives an input from the Aircraft Status which is a boolean input that defines if the aircraft is in the air (AS = on) or the aircraft is on the ground (AS = off).

SP The *DC* receives an input concerning the status of the Safety Pin which is a boolean input. If the Safety Pin is in place (SP = on) then it is not possible for DU to dispense flares. The Safety Pin has to be removed (SP = off) before the DU can dispense flares.

DS The *DC* receives an input message *con* from the *DS* which contains the flare selection and whether to release the flare or not. The message is sent at a rate of 20 messages per second. The *DS* is responsible for controlling and orchestrating all the defensive aids on the aircraft.

DU The *DU* receives a *fire* and a *sel* input from the *DC*. The *sel* input selects which flare type to release. The *fire* input commands the *DU* to release a flare if active (i.e. *fire* discrete set to on).

DC The *DC* will stop the *DU* from dispensing flares by setting the *fire* discrete to off whenever its inputs Pilot = off or *AS* = off or SP = on. The *DC* will select which flare is to be dispensed by setting the *sel* input to the *DU*, the value of *sel* is derived from the *con* message received from the *DS*. The *DC* will command the *DU* to release a flare by setting its *fire* discrete to on as long as the other inputs to the *DC* support this.

This sub-system theory of operation and analysis work was used as the basis for producing the Equipment Specifications (ES), the final Interface Requirements Specification (IRS) and the initial allocation of requirements (ASR) to software and/or hardware. The requirement was refined from *RDC* to *Rdc* by this work. The requirement *Rdc* is presented below.

Ra The *DS* shall command which flare is selected using a field in its *con* message issued to the *DC*. The *DC* shall obtain the selected flare information from this field in the *con* message, and use it in its *sel* message to the *DU* to control the flare selection in the *DU*.

Rb The *DS* shall command the *DC* to issue a *fire* command in its *con* message. This shall be the only way in which a flare can be released.

Rc The *DC* shall cause a flare to be released by issuing a *fire* command to the *DU*, which will fire the selected flare.

Rd The *DC* shall only issue a fire command if its interlocks are satisfied, i.e. aircraft is in air ($AS = on$), *SP* safety pin has been removed ($SP = off$) and *Pilot* has issued an allow a release command ($Pilot = on$).

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

Therefore, the overall requirement is $Rdc = Ra \wedge Rb \wedge Rc \wedge Rd \wedge RS$.

One of the IPDP safety tasks requires that a safety analysis is applied to the combination of the sub-system theory of operation and the system block diagram shown in Figure A.1. The results of this analysis being recorded in the Preliminary Hazard Analysis Report (PHAR) – which has a document structure that was introduced to improve the safety management in IPDP. The abstract nature of the specification (and missing timing details etc.) indicate that an FFA is the most appropriate form of analysis to use in this case (design instability and not enough information to support a HAZOPS). The main FFA guide words [90] are as follows:

- Function not provided when required;
- Function provided when not intended;
- Function provided incorrectly.

These were applied to each of the descriptive blocks in turn. For example, consider the descriptive text for the *DS* given above. This description has three parts as shown below:

- The *DC* receives an input message *con* from the *DS* which contains the flare selection

Table A.1: FFA of the *DS* to *DC* Serial Link

FFA Guide Word	Potential Failure Mode
Select: Function not provided	Wrong flare type selected for release
Select: Function when not intended	Wrong flare type selected for release
Select: Function incorrect	Wrong flare type selected for release
Fire: Function not provided	Flare not released when required
Fire: Function when not intended	Flare release too early or late
Fire: Function incorrect	Flare released inadvertently
Rate: Function not provided	No flare release possible
Rate: Function when not intended	Flare release too early or late
Rate: Function incorrect	Flare release too early or late

- and whether to release the flare or not.

- The message is sent at a rate of 20 messages per second.

Applying the guide words to these three terms (Select, Fire and Rate) yields Table A.1.

The FFA from Table A.1 identified that a function incorrect on the Fire message could result in "Flare released inadvertently" which corresponds to the identified hazards **H1** and **H2**. The function incorrect could be caused by a systematic problem in the *DS* or by errors occurring during the transmission of the *con* message. The interface between the *DS* and the *DC* was not fully explored during the initial IPDP work and this resulted in only a simple parity check mechanism being proposed for the serial link. However, serial link integrity issues identified on another Company project resulted in a specific serial link safety investigation being applied to the *DC* system. This investigated the behaviour of the *DS*, its environment and the serial link in more depth, and identified that a much stronger cyclic redundancy check (CRC) algorithm was required to obtain the desired level of integrity. This information was associated with the *DS* environment, but had been omitted from the FFA work. It was also noted that a HAZOPS (due to the serial link expertise contained in the HAZOPS team) rather than a FFA (produced by a single safety engineer) would have identified the serial link issue earlier.

All these artefacts produced during the Dem/Val phase were checked as part of the IPDP SDR checkpoint gate review. This included confirmation that each requirement had an appropriate verification method assigned to it, and checking that any issues identified by the safety analysis had been investigated and resolved.

A related issue is that IPDP allows projects to "tailor out" (i.e. remove) tasks that are not thought to be directly relevant. In the past over-zealous (and optimistic!) project managers have tailored out some of the safety analysis tasks. The rationale for this was the misunderstanding that there were already other safety tasks, without realising that each IPDP safety task has a specific purpose. This was another factor in creating the PHAR to manage the early safety analysis activities and provide a framework to ensure that all the necessary analysis work was undertaken. As noted above, this does ensure that the necessary safety analysis work is undertaken, but it does not ensure its technical quality, completeness or consistency.

A.4 EMD: Preliminary Design Phase for the DC Case Study

The activities and tasks associated with the EMD Preliminary Design Phase, and how it makes use of the validated outputs from the SDR (namely the ES, ISR and ASR documents) are described in section 3.4. The *DC* system was put through this process and as an example of the work involved, the detailed architectural analysis work is summarised below.

The architectural analysis identified an expansion of the *DC* into the three sub-components shown in Fig. A.2. The *DC* system architecture consists of three components, the Safety Controller (*SC*), the Decoy Micro-controller (*DM*, shown further expanded in Fig. A.2(b)) and the Interlock Input (*II*), as shown in Fig. A.2(a). This choice of architecture was selected because it was based on an existing prototype

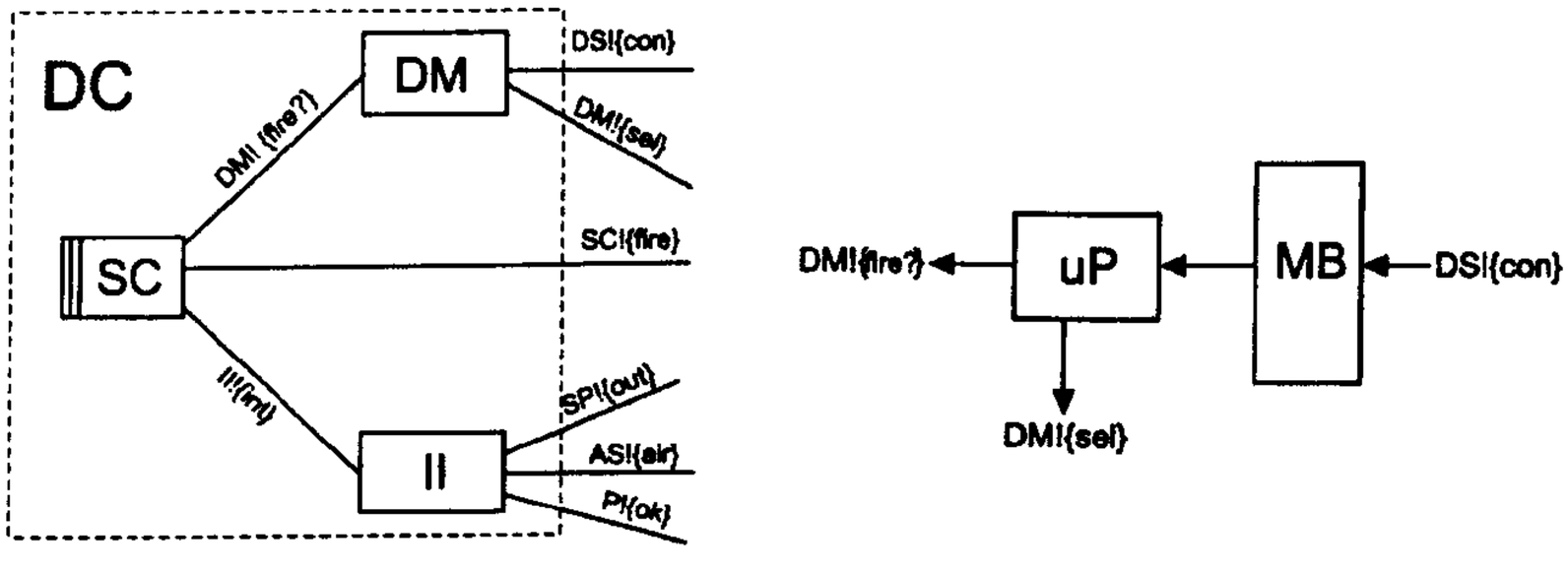

Figure A.2: (a) *DC* Architecture and (b) *DM* Internal Architecture

that was known to be capable of satisfying the functional requirement. Further, it is typical of industrial safety design strategies that attempt to minimise the number and extent of the safety related functions by localising them to simple, distinct blocks.

Briefly, component *II* collects together the interlock inputs and passes their status to *SC* (*int*). Component *DM* is a microcontroller used to decode messages from the Defence System (*con*), and when appropriate to issue the fire command request to the *SC* (via *fire*?). The Message Buffer (*MB*, in Fig. A.2(b)) holds the received control message *con* from the *DS*. The micro-controller *uP* decodes this message to extract: a) the fire command request status (*fire*?) sent to the *SC*, and b) the selected flare type (*sel*) sent to the *DU*. The *SC*, the component to be designed, is intended as a simple block that handles the safety critical elements of the interlocking. *SC* is, therefore, expected to relate an active *fire*? request to the *DU* (through phenomenon *fire*) if the interlocks are satisfied.

The introduction of the *DC* system architecture affects the high level Customer requirement *Rdc* and this needs to be refined such that terms in *DC* are replaced by terms in *DM*, *SC* and *II* as appropriate. This is a non-trivial transformation and is carefully checked by independent expert review. The result is a new requirement statement

$$\mathbf{R' = R'a \,\&\, R'b \,\&\, R'c \,\&\, R'd \,\&\, R'S}$$

in which **R'a** is **Ra** with *DC* replaced by *DM*; similarly for **R'c** and **R'd**, *mutandis*

mutatis. The most significant change occurs for **R'b** (changes in bold):

R'b The *DS* shall command the *DM* to issue a *fire*? command in its *con* message. **The *DM* will request the *SC* to send the *fire* command.** This shall be the only way in which a flare can be released.

During Preliminary Design a HAZOPS is conducted to confirm the system model is a suitable basis for further development in terms of its ability to satisfy the identified safety requirement placed on it. At this point the required supporting design information is available and the design is more stable – so HAZOPS is cost-effective to apply at this time.

Appendix B

More PPTs Applied to the *DC* Case Study

B.1 PPT Applied to the *AS* Domain

PS4: Application of the PPT for AS Domain Removal to problem P_{Red1} to form P_{Red2}

JUSTIFICATION *J*: The *AS* domain removal is based on the process given in section 5.1.2. The *AS* domain is adjacent to the requirement *R*1. The *AS* domain control phenomena *air* maps directly to its output behaviour phenomena which is also *air*. Therefore *air* will appear in the reference part of the requirement phenomena both before and after the transformation. The requirement *R*1 is as detailed in step PS3 on page 115. Inspection of *R*1 indicates that only **R1d** involves a reference to *air*.

Consider the transformation from **R1d** into **R2d** which involves the phenomena *air* as noted above.

R1d : aircraft is in air (*air* = *yes*).

R2d : *II* observes input *air* = *yes*.

AS Domain Description : The in-air indicator is obtained from the weight on wheels

and landing gear up indications: if the landing gear is up and there is no weight on the wheels then the aircraft is assumed to be in the air. The landing gear is detected as being up by a number of sensor switches. The switches use a multi-pole arrangement of appropriately selected "Normally open/Normally closed" contacts. This imbues an error detection capability that is used to achieve very good failure rates, well within the required margins. The setting of the in-air indicator is determined by the AS domain and is output using the phenomena air, where $air = yes$ means the aircraft is in the air, whilst $air = no$ means the aircraft is on the ground.

The $A_2 = Assumption(air, air)$ is given by the *AS Domain Description*. Inspection of $R2d$ in conjunction with the *AS Domain Description* shows they imply $R1d$ as required.

Therefore the PPT application transforms the problem from P_{Red1} into P_{Red2}, and $R1$ is transformed into $R2$ as follows:

R2a The DS shall command which flare is selected using a field in its *con* message issued to the DM. The DM shall obtain the selected flare information from this field in the *con* message, and use it in its *sel* message to the DU to control the flare selection in the DU.

R2b The DS shall command the DM to issue a *fire*? command in its *con* message. The DM will request the SC to send the *fire* command. This shall be the only way in which a flare can be released.

R2c The SC shall cause a flare to be released by issuing a *fire* command to the DU, which will fire the selected flare.

R2d The SC shall only issue a fire command if its interlocks, represented by *int*, are satisfied, i.e. II **observes input** $air = yes$, SP safety pin has been removed ($out = yes$) and II observes pilot input of $ok = yes$.

RS The DC shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

The constraint being relied upon is that $air = yes$ means the aircraft is in the air – see claim below.

The domain AS is no longer referenced in the model and can be removed.

The software problem is transformed to P_{Red2}:

$$P_{Red2}: \begin{array}{l} DS(con)^{con},\ DU(fire, sel)_{fire,sel},\ SP(out)^{out}, \\ II(ok)^{int}_{ok,air,out},\ DM^{sel,fire?}_{con},\ SC^{fire}_{int,fire?} \end{array} \vdash R2^{fire,sel}_{con,out,air,ok} \wedge A_1 \wedge A_2$$

CONCERNS: The control input phenomena *air* is the same as the domain behaviour output phenomena for this reference-type problem progression. This can be a symptom of over-specification.

CLAIM: *The AS intention with respect to interlocking flare release is represented by the phenomena air, so no over-specification requirement issue arises.*

ARGUMENT & EVIDENCE: This is covered by the *AS Domain Description* and the validation as described above.

CLAIM: *The in-air status of the aircraft is represented by the phenomena air.*

ARGUMENT & EVIDENCE: This is covered by the *AS Domain Description.*

PHENOMENA: None

Note that often the useful behaviour assumptions are omitted from the Problem description, but they are always a necessary part of the reduction process.

B.2 PPT Applied to the *SP* Domain

PS5: Application of the PPT for SP Domain Removal to problem P_{Red2} to form P_{Red3}

JUSTIFICATION J: The *SP* domain removal is based on the process given in section

5.1.2. The *SP* domain is adjacent to the requirement *R*2. The *SP* domain control phenomena *out* maps directly to its output behaviour phenomena which is also ***out***. Therefore *out* will appear in the reference part of the requirement phenomena both before and after the transformation. The requirement *R*2 is as detailed in step PS4 on page 235. Inspection of *R*2 indicates that only **R2d** involves a reference to *out*.

Consider the transformation from **R2d** into **R3d** which involves the phenomena ***out*** as noted above.

*R*2*d* : *SP* safety pin has been removed (*out* = *yes*).

*R*3*d* : *II* observes input *out* = *yes*.

SP Domain Description : The safety pin status is obtained from sensor switches. The switches use a multi-pole arrangement of appropriately selected "Normally open/Normally closed" contacts. This imbues an error detection capability that is used to achieve very good failure rates, well within the required margins. The setting of the safety pin status is determined by the *SP* domain and is output using the phenomena *out*, where *out* = *yes* means the safety pin has been removed, whilst *out* = *no* means the safety pin is present (and inhibiting flare release).

The $A_3 = Assumption(out, out)$ is given by the *SP Domain Description*. Inspection of *R*3*d* in conjunction with the *SP Domain Description* shows they imply *R*2*d* as required.

Therefore the PPT application transforms the problem from P_{Red2} into P_{Red3}, and *R*2 is transformed into *R*3 as follows:

R3a The *DS* shall command which flare is selected using a field in its *con* message issued to the *DM*. The *DM* shall obtain the selected flare information from this field in the *con* message, and use it in its *sel* message to the *DU* to control the flare selection in the *DU*.

R3b The *DS* shall command the *DM* to issue a *fire*? command in its *con* message. The *DM* will request the *SC* to send the *fire* command. This shall be the only way in which a flare can be released.

R3c The *SC* shall cause a flare to be released by issuing a ***fire*** **command to the** *DU*, which will fire the selected flare.

R3d The *SC* shall only issue a fire command if its interlocks, represented by ***int***, **are** satisfied, i.e. *II* observes input $air = yes$, *II* **observes input** $out = yes$ **and** *II* observes pilot input of $ok = yes$.

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

The constraint being relied upon is that $out = yes$ means the safety pin has been removed – see claim below.

The domain *SP* is no longer referenced in the model and can be removed.

The software problem is transformed to P_{Red3}:

$$P_{Red3}: \quad DS(con)^{con},\ DU(fire, sel)_{fire,sel},\ II(ok)^{int}_{ok,air,out},\ DM^{sel,fire?}_{con},\ SC^{fire}_{int,fire?} \vdash R3^{fire,sel}_{con,out,air,ok} \wedge A_1 \wedge A_2 \wedge A_3$$

CONCERNS: The control input phenomena *out* is the same as the domain behaviour output phenomena for this reference-type problem progression. This can be a symptom of over-specification.

CLAIM: *The SP intention with respect to interlocking flare release is represented by the phenomena out, so no over-specification requirement issue arises.*

ARGUMENT & EVIDENCE: This is covered by the *SP Domain Description* and the validation as described above.

CLAIM: *The safety pin status is represented by the phenomena out.*

ARGUMENT & EVIDENCE: This is covered by the *SP Domain Description.*

PHENOMENA: None

B.3 PPT Applied to the *DS* Domain

PS6: Application of the PPT for DS Domain Removal to problem

P_{Red3} to form P_{Red}

JUSTIFICATION *J*: The *DS* domain removal is based on the process given in section 5.1.2. The *DS* domain is adjacent to the requirement *R*3. The *DS* domain control phenomena *con* maps directly to its output behaviour phenomena which is also *con*. Therefore *con* will appear in the reference part of the requirement phenomena both before and after the transformation. The requirement *R*3 is as detailed in step PS5 on page 237. Inspection of *R*3 indicates that both **R3a** and **R3b** involve references to *con*.

Consider the transformation from **R3a** into **R4a** which involves the phenomena *con* as noted above.

*R*3*a* : The *DS* shall command which flare is selected using a field in its *con* message.

*R*4*a* : The *DM* shall decide which flare to select by observing a field in the *con* message it receives.

A similar arrangement can be constructed for flare release for covering the transformation from **R3b** into **R4b**.

DS Domain Description : The *DS* monitors many systems (e.g. early warning radar, GPS/INS positioning) to determine any threats to the aircraft. These systems are not pertinent to this problem so are abstracted through the statement:

> The *DS* is the defensive aids suite controller function which selects the flare type for release and if and when a flare is released in response to any threats it determines from its monitoring functions. The *DS* achieves this by formatting the flare selection and fire commands into its *con* message which it transmits to the *DM*.

The $A_4 = Assumption(con, con)$ is given by the *DS Domain Description*. Inspection

of *R4a* in conjunction with the *DS Domain Description* shows they imply *R3a* as required.

The *DM* knows the structure of the *con* message it receives from the *DS*, so it can decode the *con* message to determine the flare selection (*sel*) and fire (*fire*) components.

Therefore the PPT application transforms the problem from P_{Red3} into P_{Red}, and *R3* is transformed into *R4* as follows:

R4a The *DM* shall decide which flare to select by observing a field in the *con* message it receives. The *DM* shall obtain the selected flare information from this field in the *con* message, and use it in its *sel* message to the *DU* to control the flare selection in the *DU*.

R4b The *DM* shall observe the *con* message it receives and issue a *fire*? command if commanded. The *DM* will request the *SC* to send the *fire* command. This shall be the only way in which a flare can be released.

R4c The *SC* shall cause a flare to be released by issuing a *fire* command to the *DU*, which will fire the selected flare.

R4d The *SC* shall only issue a fire command if its interlocks, represented by *int*, are satisfied, i.e. *II* observes input *air* = *yes*, *II* observes input *out* = *yes* and *II* observes pilot input of *ok* = *yes*.

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

The domain *DS* is no longer referenced in the model and can be removed.

The REQUIREMENTS INTERPRETATION transformation associated with the PPT establishes that *con* is equivalent to (*fire*?, *sel*), this information is used in the software problem P_{Red}:

$$P_{Red}: \quad \begin{array}{l} DU(fire, sel)_{fire,sel}, \quad II(ok)^{int}_{ok,air,out}, \\ DM^{sel,fire?}_{con}, \quad SC^{fire}_{int,fire?} \quad \vdash \quad R4^{fire,sel}_{con,out,air,ok} \wedge A_1 \wedge A_2 \wedge A_3 \wedge A_4 \end{array}$$

CLAIM: *The flare selection and flare fire commands are represented by the phenomena con.*

ARGUMENT & EVIDENCE: This is covered by the *DS Domain Description.*

The following directly support the analysis tasks:

Note1: **R4a and R4b** relate the requirement phenomena *con* to the causal chain phenomena *fire*? and *sel*.

Note2: **R4d** relates the requirement phenomena *air*, *out* and *ok* to the causal chain phenomena *int*.

CLAIM: *The chosen solution architecture does not prevent the satisfaction of R.* **This claim does not hold.**

ARGUMENT & EVIDENCE: PSA (FFA, HAZOPS and FTA) was again applied to each architectural component in turn. The results are presented in section 5.2.2 and show that **the current design does not satisfy the safety requirement.**

PHENOMENA: None

Appendix C

High Integrity IPDP Safety Process

The existing successful Z-based safety critical development process has the following characteristics. Briefly, system safety properties are formally captured in Z and transcribed (rather than refined) into a detailed Z design specification, against which formal code proof using SPARK [7] is performed. An important validation step is the proof of conformance of the Z design specification against the formal Z safety properties. This process provides a formal path from the high level safety properties down to the code that implements them. However, validation occurs well into the design process and it has uncovered anomalies with the Z safety properties. That is, this safety critical development approach has uncovered anomalies in the Z safety properties rather late in the IPDP than is optimal. The anomalies uncovered ranged from a misinterpretation of the actual physical sequence (a modelling error), through to contradictory requirements. One of the aims of this case study is to show that, using the POSE/Alloy combination, those anomalies *could* have been identified and resolved earlier in the development life cycle. The ultimate goal of this work is to enhance and evolve what is already a successful safety critical development process, not to change to some novel process of unknown pedigree.

Appendix D

POSE Steps for the SMS Case Study

One of the major initial tasks in IPDP CE phase is to fully understand the problem context. The initial requirement is introduced into the problem during the CE phase, and developed through into the early part of Dem/Val. The initial system architecture is introduced during Dem/Val. The information collected through this work is collated into the step SPS1 shown below.

SPS1: Application of Initial Problem understanding to problem

P_{null}

JUSTIFICATION J_1: The justifications for the initial POSE transformations are as follows:

[Domain Interpretation, J_a] The analysis of the information sources identified that the environment consisted of the *SMS*, the *S&RE* [1] and the *Pilot* and that their interactions were as discussed above in section 5.5.1.

[1] *S&RE*: Suspension and Release Equipment. Holds the stores on the aircraft until a release or jettison is enacted.

[Requirement Interpretation, J_b] The identified requirement was based on information provided by the customer and was validated through reviews, email correspondence and meetings. It is presented as R (**Ra** to **Ri**) below.

[Solution Interpretation, J_c] This is a typical representation of an *SMS*.

The domains and their relevant properties are summarised in $P_{Initial}$ presented below, where the justification for this initial software problem is $J_1 = J_a \wedge J_b \wedge J_c$.

PHENOMENA: Phenomena and their control and sharing are as follows: The *Pilot* controls *SJ* in the *SMS* using three phenomena: (a) *p_mode*: a selection of *SJ* mode (Off, Ship or Station); (b) *p_sel*: a set of stations selected for jettison if Station mode selected; (c) *p_jet*: an *SJ* button that initiates jettison and must be held on for the duration of the jettison sequence.

The *SMS* controls the *S&RE* using the phenomena *jetv*, a vector of station positions to which jettison is to be applied. The *SMS* also takes in the aircraft in air status using the phenomena *InAir*.

The referenced requirement phenomena, *ref*, are concerned with the *Pilot*s intention to select the *SJ* mode (*p_mode*), the stations for *SJ* (*p_sel*) and requesting jettison (*p_jet*). The requirement also references the *InAir* status. The constrained requirement phenomena, *cons*, are concerned with selecting which *S&RE* stations are to be jettisoned (*jet_stat*).

Where: $cons = \{jet\_stat\}$ and $ref = \{p\_mode, p\_sel, p\_jet\}$

The initial software problem is as follows:

$$P_{Initial}: \quad \begin{array}{l} S\&RE(jet\_stat)_{jetv},\ Pilot(p\_mode, p\_sel, p\_jet)^{smode,sstat,ssj}, \\ SMS(InAir)^{jetv}_{smode,sstat,ssj} \end{array} \quad \vdash \quad R^{jet\_stat}_{p\_mode,p\_sel,p\_jet,InAir}$$

A PF problem diagram representation of $P_{Initial}$ is shown in Figure D.1.

The high level *SMS* requirement, *R*, is as follows:

Ra The *SMS* shall ensure the safe release of stores from the *S&RE* when commanded by the *Pilot* using *p_mode*, *p_sel* and *p_jet*.

Rb The *SMS* shall have a Selective Jettison (*SJ*) facility which shall allow the *Pilot*, using *p_mode*, to jettison all of the stores (Ship mode) or the jettison of selected stores (Station mode) from the *S&RE*.

Rc The *SMS* shall allow the *Pilot* to select each of the six *S&RE* stations for *SJ* using *p_sel*. In Station *SJ* mode, a station must be selected before it can be jettisoned.

Rd The *SMS* shall have an In Air lamp, which will be lit if the aircraft is in the air. *SJ* is only allowed if the aircraft is in the air.

Re The *SMS* shall have a *SJ* button to allow the *Pilot* to initiate the jettison using *p_jet*, and it must remain pressed for the duration of the jettison sequence. The following must hold prior to initiating the *SJ*:

1. *SJ* mode selected (Ship or Station).
2. Aircraft is in the air.
3. If Station mode, the appropriate stations are selected in the *SMS*.

Rf The *SMS* shall control the selection and jettison of stores from the *S&RE*, by providing the conditioned power and drive signals to correctly drive and control the *S&RE*.

Rg The *SMS* shall implement a Balance Algorithm which can remove items from an *SJ* package to ensure aircraft roll moments are contained within safe limits.

Rh It shall be possible to *SJ* a store from a single station without regard to the aircraft balance algorithm calculation.

Ri The *SJ* release sequence will be Out stores first, then Mid and then In.

CLAIM: *The interpretations are well-founded*

ARGUMENT & EVIDENCE: The choice of domains follows from the aircraft level safety analysis and the required choice of interlocks. The *S&RE*, is an existing component

of the avionics system, with well-known, validated properties. The *Pilot* is trained to follow protocols rigorously. The *SMS* is to be designed, but the Company are domain experts for this type of system.

The customer functional requirement (**Ra** to **Ri**) was provided as an input to the developer team, and was validated with the customer.

CLAIM: *The architecture is feasible*

ARGUMENT & EVIDENCE: The selected overall system architecture has been used successfully on similar *SMS* safety systems.

CLAIM: *Sound judgement followed*

ARGUMENT & EVIDENCE: Development based on IPDP used successfully on many previous developments and enhanced with POSE to mitigate some known IPDP issues. Therefore it follows normal and not radical design concepts as defined in [131].

CLAIM: *The S&RE component sub-systems have sufficient reliability to allow the overall system to satisfy its objectives*

ARGUMENT & EVIDENCE: The reliability claims are justifiable since the S&RE components are part of an established successful design, but are not included in detail for reasons of brevity.

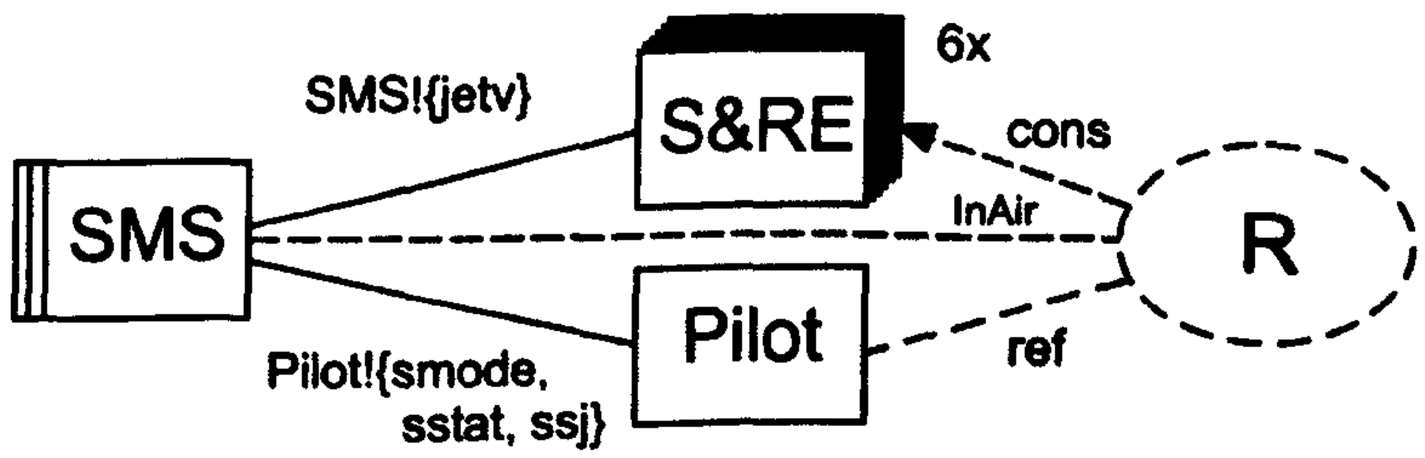

Figure D.1: *SMS* Initial Problem, $P_{Initial}$

The next stage is to investigate candidate system architectures. The most promising candidate is shown in step SPS2 presented below.

SPS2: Application of SOLUTION INTERPRETATION AND EXPANSION *to problem* $P_{Initial}$

JUSTIFICATION J_2: The expanded *SMS* solution architecture consists of three types of domain one of Safety Monitor (*SM*), six of Store Unit (*SU*) and one of Selective Jettison Panel (*SP*).

The *SP* shown in Figure 5.7 indicates the aircraft has six release stations—corresponding to Outer, Middle and Inner positions on each of the Starboard and Port wings. The pilot (*Pilot*) controls store jettison via the *SP*, using (a) six selection switches, one for each station, (b) a three position *SJ* mode selection switch, (c) an *InAir* indicator lamp and (d) the *SJ* button which initiates the jettison. The *InAir* indicator is fed from another system (not developed further here).

Each *SU* is connected to one of the six *S&RE* units and provides the conditioned power signal to release the store attached to the *S&RE*. Each *SU* is controlled by the *SM*.

The *SM* is the control function for the *SMS*. It reads in the *SP* status for mode, selected stations, the *SJ* initiation button, and the *InAir* status.

The justification for this (architecture) solution expansion transformation is that this system architecture has been successfully used on a number of other *SMS* developments and is known to be capable of satisfying the functional requirement.

PHENOMENA: The new phenomena introduced by the architecture are:

invec – Status of the *SJ* mode, station selection, *InAir* and *SJ* button inputs

outv – A vector of discrete release/jettison control signals, one to each of the six *SU*s.

The expanded software problem is as follows:

$$P_{Exp}: \quad S\&RE(jet\_stat)_{jetv},\ Pilot(p\_mode, p\_sel, p\_jet)^{smode,sstat,ssj},$$
$$SP(InAir)^{invec}_{smode,sstat,ssj},\ SU^{jetv}_{outv},\ SM^{outv}_{invec} \quad \vdash \quad R'^{jet\_stat}_{p\_mode,p\_sel,p\_jet,InAir}$$

A PF problem diagram representation of P_{Exp} is shown in Figure D.2.

The transformed requirement R' is similar to R apart from the changes introduced by expanding the domains from *SMS* to *SP*, *SM* and *SU*. For example all references to the *Pilot* in R controlling the *SMS* are replaced with references to *SP*, as the *SP* is the means by which the *Pilot* interfaces to the stores management system. The biggest change is to **R'f** where *SMS* is replaced by references to *SM* and its control of the *SU*s.

The updated requirement R' is as follows:

R'a The *SM* shall ensure the safe release of stores from the *S&RE* when commanded by the *Pilot* using *p_mode*, *p_sel* and *p_jet*.

R'b The *SM* shall have a Selective Jettison (*SJ*) facility which shall allow the *Pilot*, using *p_mode*, to jettison all of the stores (Ship mode) or the jettison of selected stores (Station mode) from the *S&RE*.

R'c The *SP* shall allow the *Pilot* to select each of the six *S&RE* stations for *SJ* using *p_sel*. In Station *SJ* mode, a station must be selected before it can be jettisoned.

R'd The *SP* shall have an In Air lamp, which will be lit if the aircraft is in the air. *SJ* is only allowed if the aircraft is in the air.

R'e The *SP* shall have a *SJ* button to allow the *Pilot* to initiate the jettison using *p_jet*, and it must remain pressed for the duration of the jettison sequence. The following must hold prior to initiating the *SJ*:

1. *SJ* mode selected (Ship or Station).
2. Aircraft is in the air.
3. If Station mode, the appropriate stations are selected on the *SP*.

R'f The *SM* shall control the selection and jettison of stores from the *S&RE*, by controlling the *SU*s to provide conditioned power and drive signals to correctly drive and control the *S&RE*s.

R'g The *SM* shall implement a Balance Algorithm which can remove items from an *SJ* package to ensure aircraft roll moments are contained within safe limits.

R'h It shall be possible to *SJ* a store from a single station without regard to the aircraft balance algorithm calculation.

R'i The *SJ* release sequence will be Out stores first, then Mid and then In.

CLAIM: *Justification for selection of the solution architecture*

ARGUMENT & EVIDENCE: This system architecture has been successfully used on a number of other *SMS* developments and is known to be capable of satisfying the functional requirement.

CLAIM: *The choice of candidate solution architecture exhibits sound safety engineering judgement*

ARGUMENT & EVIDENCE: The architecture is chosen to maximise integrity. The jettison status information from the *SP* to the *SM* is transferred by the serial link message *invec* which includes a cyclic redundancy check (CRC) for error detection. The control information from the *SM* to the *SU*s is transferred as a vector of discrete signals *outv* (one for each station).

CLAIM: SJ *push-button switch on the SP is reliable*

ARGUMENT & EVIDENCE: The *SJ* push-button switch indication is obtained from a number of sensor switches. The switches use a multi-pole arrangement of appropriately selected "Normally open/Normally closed" contacts. This imbues an error detection capability that is used to achieve very good failure rates, well within the required margins.

Similar reliability claims can be made for the mode *SJ* mode switch and the station selection switches on the *SP* – not shown for brevity.

CLAIM: *The chosen solution architecture does not prevent the satisfaction of R.* **This claim is not yet supported - deferred until PSA after PPT.**

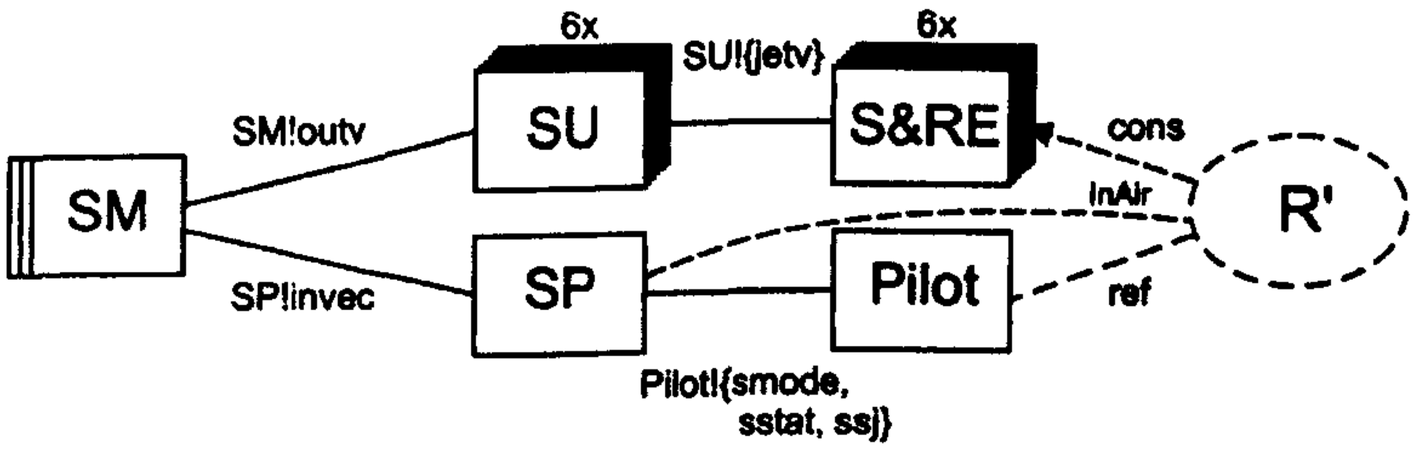

Figure D.2: *SMS* Expanded Problem, P_{Exp}

First application of the PPT to simplify P_{Exp} by removing the *Pilot* domain.

SPS3: Application of the PPT for Pilot Domain Removal to problem P_{Exp} to form P_{Red1}

INCLUDES: None

JUSTIFICATION *J*: The *Pilot* domain removal is based on the process given in section 5.1.2. The *Pilot* domain is adjacent to the requirement R'. The *Pilot* domain is a reference-type. The *Pilot* domain *effect* is to control the jettison function using the mode, station selection and *SJ* button on the *SP*. In PF terms the *Pilot* is a biddable domain. That is, although the *Pilot* is trained to follow protocol rigorously, there is a chance a mistake could be made and any analysis has to take due account of this – this information is stored as part of the justification for the removal.

Inspection of R' in SPS2 indicates that **R'a, R'b, R'c, and R'e** involve a reference to *Pilot*. These will be updated in $R1$.

The *Pilot effect* is controlling the *SP* using the requirement phenomena ***p_mode***, ***p_sel*** and *p_jet*, in the causal chain these respectively relate directly to the phenomena *smode*, *sstat*, *ssj*.

Consider the transformation from **R'a** involving "commanded by the *Pilot*". Commanded by the *Pilot* corresponds to appropriately setting the phenomena *p_mode*, *p_sel*, and *p_jet* on the *SP*. This is achieved through the phenomena *smode*, *sstat* and *ssj*. Therefore **R1a** has the form shown below.

Consider the transformation from **R'b** involving "*Pilot* using *p_mode*" to jettison all stores or selected stores. In the causal chain the *p_mode effect* is related to *smode*. Therefore the *R1b* text is rewritten as "The *SM* shall have a Selective Jettison (*SJ*) facility which will allow the jettison of all stores (Ship mode) or the jettison of selected stores (Station mode) from the *S&RE* in accordance with *smode*".

Consider the transformation from **R'c** involving "allow the *Pilot* to select each of the six *S&RE* stations for *SJ*". The *Pilot* selects stations through *p_sel*, in the causal chain this *effect* is related to *sstat*. Therefore in *R1c* the text is re-written as "allows each of the six *S&RE* stations to be selected for *SJ* in accordance with the *sstat*".

Consider the transformation from **R'e** involving "to allow the *Pilot* to initiate the jettison". The *Pilot* initiates jettison through *p_jet*, in the causal chain this *effect* is related to *ssj*. Therefore in **R1e** the text is re-written as "that allows jettison to be initiated in accordance with *ssj*". In addition other terms in **R'e** are updated as appropriate to be in accordance with *smode* and *sstat* to form **R1e**.

These requirement transformations do not require additional information not already in the interface nor do they refer to future events. However they do rely on information from the domain being removed being "saved" as part of the assumption that justifies the transformation. That is, we have that the *Pilot* useful behaviour relation (saved as part of the assumption) in conjunction with R' leads to $R1$. Therefore this is an example of an *Environment Constraint* requirement.

Therefore the PPT application transforms the problem from P_{Exp} into P_{Red1}, and R' is transformed into $R1$ as follows (where items in bold denote the areas that have been

changed):

R1a The *SM* shall ensure the safe release of stores from the *S&RE* when **commanded by the appropriate settings of** *smode*, *sstat*, **and** *ssj*.

R1b The *SM* shall have a Selective Jettison (*SJ*) facility which will allow the jettison of all stores (Ship mode) or the jettison of selected stores (Station mode) from the *S&RE* **in accordance with** *smode*.

R1c The *SP* shall **allows each of the six** *S&RE* **stations to be selected for** ***SJ*** **in accordance with the** *sstat*. In Station *SJ* mode, a station must be selected before it can be jettisoned.

R1d The *SP* shall have an In Air lamp, which will be lit if the aircraft is in the air. *SJ* is only allowed if the aircraft is in the air.

R1e The *SP* shall have a *SJ* button **that allows jettison to be initiated in accordance with** *ssj* and it must remain pressed for the duration of the jettison sequence. The following must hold prior to initiating the *SJ*:

1. *SJ* mode selected (Ship or Station) **in accordance with** *smode*.
2. Aircraft is in the air.
3. If Station mode, the appropriate stations are selected on the *SP* **in accordance with** *sstat*.

R1f The *SM* shall control the selection and jettison of stores from the *S&RE*, by controlling the *SU*s to provide conditioned power and drive signals to correctly drive and control the *S&RE*s.

R1g The *SM* shall implement a Balance Algorithm which can remove items from an *SJ* package to ensure aircraft roll moments are contained within safe limits.

R1h It shall be possible to *SJ* a store from a single station without regard to the aircraft balance algorithm calculation.

R1i The *SJ* release sequence will be Out stores first, then Mid and then In.

The constraint being relied upon is that the *Pilot effects p_mode*, *p_sel* and ***p_jet*** correspond directly to the phenomena *smode*, *sstat* and *ssj* respectively. This is the useful behaviour assumption for this domain removal transformation step.

The domain *Pilot* is no longer referenced in the model and can be removed.

The problem is transformed from P_{Exp} into P_{Red1} which is shown in POSE sequent form as follows:

$$P_{Red1}: \quad \begin{array}{l} S\&RE(jet\_stat)_{jetv}, SP(smode, sstat, ssj, InAir)^{invec}, \\ SU^{jetv}_{outv}, \quad SM^{outv}_{invec} \vdash R1^{jet\_stat}_{smode,sstat,ssj,InAir} \end{array}$$

Note that the usual convention is not to show the useful behaviour assumption explicitly with the software problem and this is adopted for P_{Red1} and in the other domain removals that follow.

CLAIM: *The pilot's intention with respect to performing jettison is represented by the phenomena smode, sstat and ssj.*

ARGUMENT & EVIDENCE: This is covered by the justification given above.

PHENOMENA: None

The removal of the $S\&RE$ domains is typified by the following problem transformation step.

SPS4: Application of the PPT for S&RE Domain Removal to problem P_{Red1} to form P_{Red}

JUSTIFICATION *J*: The $S\&RE$ domains removal is based on the process given in section 5.1.2. The $S\&RE$ domains are adjacent to the requirement $R1$. The $S\&RE$ domains are constrain-type PPTs. The $S\&RE$ domains *effect* is to release the selected

stores. The phenomena *jetv* is a vector quantity that causes this store release *effect* for each of the selected stations. This is achieved by providing the selected $S\&RE$ stations with conditioned drive power.

The requirement $R1$ constrains the behaviour of the $S\&RE$ through the phenomena *jet_stat*. In the domain removal the requirement will be transformed from $R1$ to $R2$. Inspection of $R1$ in SPS3 indicates that **R1a, R1b, R1c, and R1f** involve a reference to $S\&RE$. These will be updated in $R2$.

Consider the transformation from **R1a** involving "safe release of stores from the $S\&RE$". In this PPT the $S\&RE$ *effect* of store release is replaced by the SU phenomena *jetv* that selects stores to receive the conditioned release drive signals. Therefore **R2a** has the form "safe selection of stores through the SU phenomena *jetv*.".

Consider the transformation from **R1b** involving "will allow the jettison of all stores (Ship mode) or the jettison of selected stores (Station mode) from the $S\&RE$ in accordance with *smode*". In this PPT the $S\&RE$ *effect* of store release is replaced by the SU phenomena *jetv* that selects stores to receive the conditioned release drive signals. Therefore **R2b** has the form "will select all stores for jettison in *jetv* (Ship mode) or select specific stores for jettison in *jetv* (Station mode) from the SU in accordance with *smode*. ".

Consider the transformation from **R1c** involving "allows each of the six $S\&RE$ stations to be selected for SJ". In this PPT the $S\&RE$ *effect* of store release is replaced by the SU phenomena *jetv* that selects stores to receive the conditioned release drive signals. Therefore **R2c** has the form "allow each of the six station selection positions in *jetv* to be selected for SJ".

Consider the transformation from **R1f** involving "the selection and jettison of stores from the $S\&RE$" and "to correctly drive and control the $S\&RE$". In this PPT the $S\&RE$ *effect* of store release is replaced by the SU phenomena *jetv* that selects stores to receive the conditioned release drive signals. Therefore **R2f** has the form "the selection of stores for jettison in the *jetv* from the SU" and "through its output phenomena *jetv*" and adding the term "correctly" earlier in the phrase.

These requirement transformations do not require additional information not already in the interface nor do they refer to future events. However they do rely on information from the domain being removed being "saved" as part of the assumption that justifies the transformation. That is, we have that $S\&RE, R2 \vdash R1$ which states that $R2$ requires assistance from the $S\&RE$ domain environment to satisfy the higher level requirement $R1$. Therefore this is an example of an *Environment Constraint* requirement.

Therefore the PPT application transforms the problem from P_{Red1} into P_{Red}, and $R1$ is transformed into $R2$ as follows (where items in bold denote the areas that have been changed):

R2a The *SM* shall ensure the **safe selection of stores through the** *SU* **phenomena** *jetv* when commanded by the appropriate settings of *smode*, *sstat*, **and** *ssj*.

R2b The *SM* shall have a Selective Jettison (*SJ*) facility which **will select all stores for jettison in** *jetv* **(Ship mode) or select specific stores for jettison in** *jetv* **(Station mode) from the** *SU* in accordance with *smode*.

R2c The *SP* shall **allow each of the six station selection positions in** *jetv* **to be selected for *SJ*** in accordance with the *sstat*. In Station *SJ* mode, a station must be selected before it can be jettisoned.

R2d The *SP* shall have an In Air lamp, which will be lit if the aircraft is in the air. *SJ* is only allowed if the aircraft is in the air.

R2e The *SP* shall have a *SJ* button that allows jettison to be initiated in accordance with *ssj* and it must remain pressed for the duration of the jettison sequence. The following must hold prior to initiating the *SJ*:

1. *SJ* mode selected (Ship or Station) in accordance with *smode*.
2. Aircraft is in the air.
3. If Station mode, the appropriate stations are selected on the *SP* in accordance with *sstat*.

R2f The *SM* shall control **the selection of stores for jettison in the *jetv* from the *SU***, by controlling the *SU*s to **correctly** provide conditioned power and drive signals **through its output phenomena *jetv*.**

R2g The *SM* shall implement a Balance Algorithm which can remove items from an *SJ* package to ensure aircraft roll moments are contained within safe limits.

R2h It shall be possible to *SJ* a store from a single station without regard to the aircraft balance algorithm calculation.

R2i The *SJ* release sequence will be Out stores first, then Mid and then In.

The software problem model after removing the *S&RE* domains is given by:

$$P_{Red}: \quad SP(smode, sstat, ssj, InAir)^{invec}, SU(jetv)_{outv}, SM^{outv}_{invec} \vdash R2^{jetv}_{smode,sstat,ssj,InAir}$$

The useful behaviour assumption associated with this domain removal is based on the relationship between the *S&RE effect* of store release that is replaced by the *SU* phenomena *jetv* that selects stores to receive the conditioned release drive signals. As noted in SPS3, this presentation follows the convention of not showing the useful behaviour assumption with the software problem model – given by P_{Red}

Reliability and other concerns and associated claim/evidence to mitigate are not presented in detail.

Appendix E

POSE/Alloy Model of the SMS Case Study

```
module examples/DPM/smsystem3

// Second (improved) version of model, with Balance determined by overall sequence as required

open util/ordering[SMState] as sms
open Alloy_Files/Alloy4/NatNumbers as nat

// Define the SM System Objects
abstract sig SJMode {}
one sig  off, ship, stat extends SJMode {}

abstract sig StatSel {}
one sig  s0, s1, s2, s3, s4, s5, s6, s7  extends StatSel {}

abstract sig SSet {}
one sig son, soff extends SSet {}

abstract sig InAir {}
one sig air, grd extends InAir {}

abstract sig SPul {}
one sig p0, p01, p10, p2 extends SPul {}

abstract sig BAL {}
one sig nob, no_bal, is_bal extends BAL {}

// Define the SM State
sig SMState {
ntime   : Natural,
mode    : SJMode,
sels    : StatSel,
sj      : SSet,
fl      : InAir,
relp    : SSet ,
pulses  : SPul,
balan   : BAL,
cnt     : Natural } {}

// Set up the ordering of SMState
pred init (sm : SMState) {sm.mode = off && sm.sels = s0 && sm.sj = soff && sm.fl = grd &&
sm.ntime = Zero && sm.relp = soff && sm.pulses = p0 && sm.balan = nob && sm.cnt = Zero}
```

```
fact cnt3 { all sm : SMState | sm.cnt = Zero or sm.cnt = One or sm.cnt = Two or sm.cnt = Three}

fact traces {
   init[sms/first[]]
   all sm : SMState - sms/last[] | let sm' = sms/next[sm] | some m : SJMode, sel :  StatSel,
      s : SSet, f : InAir |  ( Stim[sm,sm'] && SMfun[sm, sm', m, sel, s ,f] )  }


// #### DEFINE OPERATIONS #####

// System Time Clock
pred Stim(sm, sm' : SMState) {sm'.ntime = nat/inc[sm.ntime]}


// Input Vector
pred InVec(smtim, systim : Natural, inmod, mode : SJMode, insel, sel : StatSel, insj, sj : SSet,
infl, fl : InAir ) {smtim = systim && inmod = mode && insel = sel &&  insj = sj  && infl = fl}


// Operation SM Function
pred SMfun [sm, sm' : SMState, m : SJMode, sel : StatSel, s : SSet, f : InAir] {
(sm'.mode = m  && sm'.sj = s && sm'.fl = f && sm'.sels = sel)  &&
// ### If valid start of SJ sequence .......
      ( ( sm.mode  = stat &&  sm.sels != s0 && sm.sj = soff && sm.fl = air && sm.relp = soff &&
     m = stat && sel = sm.sels && s = son && f = air)
=> sm'.relp = son  && sm'.cnt = One && pulTab[sm'] && balance[sm']
         else
// ### If valid continuation of SJ sequence .........
         ( ( sm.mode  = stat && sm.sels != s0 &&  sm.sj = son && sm.fl = air && sm.relp = son &&
       m = stat && sel = sm.sels && s = son && f = air)
   => sm'.relp = son && sm'.cnt = incnt[sm.cnt] && pulTab[sm']  && balance[sm']
           else
/// ### Otherwise, reset to safe state.
           sm'.relp = soff && sm'.cnt = Zero && sm'.pulses = p0  &&  sm'.balan = nob )
       )  }


// Function for incrementing the pulse sequence count indicator
fun incnt[n : Natural] : Natural { nat/gte[n,Three] => One else nat/inc[n] }


// Operation Station Pulses sequences
pred pulTab [sm : SMState] {
```

```
/* s1 -- Port Out & Starboard In selected for SJ */
(sm.sels = s1 && sm.cnt = One => sm.pulses = p01) &&
(sm.sels = s1 && sm.cnt = Two => sm.pulses = p0) &&
(sm.sels = s1 && sm.cnt = Three => sm.pulses = p10) &&

/* s2 -- Port Out, Starboard Out & Port Mid selected for SJ */
(sm.sels = s2 && sm.cnt = One => sm.pulses = p2) &&
(sm.sels = s2 && sm.cnt = Two => sm.pulses = p01) &&
(sm.sels = s2 && sm.cnt = Three => sm.pulses = p0) &&

/* s3 -- Starboard Out, Starboard Mid, Port In & Starboard In selected for SJ */
(sm.sels = s3 && sm.cnt = One => sm.pulses = p10) &&
(sm.sels = s3 && sm.cnt = Two => sm.pulses = p10) &&
(sm.sels = s3 && sm.cnt = Three => sm.pulses = p2) &&

/* s4 -- Starboard Mid only selected for SJ */
(sm.sels = s4 && sm.cnt = One => sm.pulses = p0) &&
(sm.sels = s4 && sm.cnt = Two => sm.pulses = p10) &&
(sm.sels = s4 && sm.cnt = Three => sm.pulses = p0) &&

/* s5 -- Port Out only selected for SJ */
(sm.sels = s5 && sm.cnt = One => sm.pulses = p01) &&
(sm.sels = s5 && sm.cnt = Two => sm.pulses = p0) &&
(sm.sels = s5 && sm.cnt = Three => sm.pulses = p0) &&

/* s6 -- Starboard Mid & Port Mid selected for SJ */
(sm.sels = s6 && sm.cnt = One => sm.pulses = p0) &&
(sm.sels = s6 && sm.cnt = Two => sm.pulses = p2) &&
(sm.sels = s6 && sm.cnt = Three => sm.pulses = p0) &&

/* s7 -- Starboard Out, Mid & In selected for SJ */
(sm.sels = s7 && sm.cnt = One => sm.pulses = p10) &&
(sm.sels = s7 && sm.cnt = Two => sm.pulses = p10) &&
(sm.sels = s7 && sm.cnt = Three => sm.pulses = p10) &&

 /* s0 -- No stores selected for SJ */
(sm.sels = s0 => sm.pulses = p0)  }


// ### Operation Balance: Improved Based on Sequence ###
pred balance[sm : SMState] { sm.relp = son &&
```

```
        (sm.sels = s4 or sm.sels = s5) => sm.balan = no_bal
         else sm.balan = is_bal  }

// ### Operation Balance: Original Based on pulses ###
//pred balance[sm : SMState] { sm.relp = son &&
// ((sm.pulses = p01  or sm.pulses = p0 or sm.pulses = p10) => sm.balan = no_bal
//     else sm.balan = is_bal ) }



pred checkp1[sm1, sm2, sm3 : SMState] {
(sm1.pulses = p01 && (sm2.pulses = p01 or sm3.pulses = p01))
or
(sm2.pulses = p01 && (sm1.pulses = p01 or sm3.pulses = p01))
or
(sm3.pulses = p01 && (sm1.pulses = p01 or sm2.pulses = p01))
                                                                }

pred checkps[sm1, sm2, sm3 : SMState] {
(sm1.pulses = p2 or sm2.pulses = p2 or sm3.pulses = p2)
or checkp1[sm1, sm2, sm3]   }


// ##### PROPERTIES #####
// Pass
assert balanceCon1 {   all sm : SMState |
sm.relp = son && sm.sels = s1     => sm.balan = is_bal }
// Fail
assert balanceCon2 {   all sm : SMState |
sm.relp = son && sm.sels = s1     => sm.balan = no_bal }

// All pass:
assert balanceCon3 {   all sm : SMState |
sm.relp = son && (sm.sels = s4 or sm.sels = s5)     => sm.balan = no_bal }

assert balanceCon4 {   all sm : SMState |
sm.relp = son && not(sm.sels = s4 or sm.sels = s5)     => sm.balan = is_bal }

assert balanceCon5 {   all sm : SMState |  (sm.pulses = p2 => sm.balan = is_bal) }
```

```
assert balanceCon6 {   all sm : SMState | all  n : Natural | all s : StatSel | sm.sels = s &&
                                    nat/gte[n,One] && nat/lte[n,Three] && sm.cnt = n && sm.relp = son &&
                                     sm.pulses = p2 => sm.balan = is_bal }


 assert balanceCon7 {   all sm1, sm2, sm3 : SMState |
(   ( sm3 = next[sm2] && sm2 = next[sm1] && sm1.cnt = One && sm2.cnt = Two && sm3.cnt = Three)
     &&
     sm1.pulses = p10  &&  (sm2.pulses = p10  or sm3.pulses = p10)   =>
    (sm1.balan = is_bal && sm2.balan = is_bal  && sm3.balan = is_bal) )
                                                 }


 assert balanceCon8 {   all sm1, sm2, sm3 : SMState |
(   ( sm3 = next[sm2] && sm2 = next[sm1] && sm1.cnt = One && sm2.cnt = Two && sm3.cnt = Three)
     &&
     sm2.pulses = p10  &&  (sm1.pulses = p10  or sm3.pulses = p10)   =>
     (sm1.balan = is_bal && sm2.balan = is_bal  && sm3.balan = is_bal) )
                                                 }


 assert balanceCon9 {   all sm1, sm2, sm3 : SMState |
(   ( sm3 = next[sm2] && sm2 = next[sm1] && sm1.cnt = One && sm2.cnt = Two && sm3.cnt = Three)
     &&
     checkp1[sm1, sm2, sm3]   =>
    (sm1.balan = is_bal && sm2.balan = is_bal  && sm3.balan = is_bal) )
                                                  }


 assert balanceCon10 {   all sm1, sm2, sm3 : SMState |
(   ( sm3 = next[sm2] && sm2 = next[sm1] && sm1.cnt = One && sm2.cnt = Two && sm3.cnt = Three)
     &&
     checkps[sm1, sm2, sm3]   =>
    (sm1.balan = is_bal && sm2.balan = is_bal  && sm3.balan = is_bal) )
                                                 }

check balanceCon1 for 52
check balanceCon2 for 3

check balanceCon3 for 52
check balanceCon4 for 52
```

```
check balanceCon5 for 8
check balanceCon6 for 6

check balanceCon7 for 6
check balanceCon8 for 6

check balanceCon9 for 6
check balanceCon10 for 16

pred show1() {}
run show1  for 52
```

Appendix F

POSE Safety Pattern Applied to the *Warning System* Case Study

This work is based on the POSE safety pattern shown in Figure 6.1, and follows the Agenda given in Table 6.1.

F.1 Initial Problem Understanding step for the *FAS*

The first part of the POSE safety pattern is Activity 1 Initial Problem understanding which is based on the format used for PS1 (*DC* case study) and SPS1 (*SMS* case study). This work uses information gained from the IPDP CE (context and requirement) and early Dem/Val (for requirement and initial architecture) phases.

WPS1: Application of Initial Problem Understanding to problem P_{null}

JUSTIFICATION J_1: **The justifications for the initial POSE transformations are:**

In the *FAS* environment, the monitored systems typically include: inertial navigation; GPS navigation; aircraft data and environment data. Failure of these systems will not prove insurmountable to the pilot. The health status of these systems is input into the *SYS* domain. In contrast, a failure in the Catastrophic System, *CS*, (part of the flight control system) could result in the loss of the aircraft. Therefore the *SYS* and the *CS* domains provide health status information to the *FAS*. If the health status information indicates a problem, then the *FAS* plays an appropriate warning message to the pilot through the pilot's headphones.

[Domain Interpretation, J_a] The analysis of the information sources identified that the significant environment components consisted of the *FAS*, the *SYS*, the *CS*, the *Pilot* and the pilot's headphones, *Speaker*.

[Requirement Interpretation, J_b] The identified requirement was based on information provided by the customer and validated through reviews, email correspondence and meetings. It is presented as *R* (**Ra** to **Rd** and *RS*) below.

[Solution Interpretation, J_c] This high-level system architecture representation of the *FAS* was based on an existing, successful design.

The domains and their relevant properties are summarised in $P_{Initial}$ presented below, where the justification for this initial software problem is $J_1 = J_a \wedge J_b \wedge J_c$.

PHENOMENA: Phenomena and their control and sharing are as follows: The *FAS* can be seen to monitor directly the status of the Catastrophic System (CS) using a discrete input *cat*. It also monitors the status of the other critical aircraft systems (*SYS*) and this is represented by the generic *sys* status message. The FAS issues warning *audio* messages to the pilot via the pilot's headphones

The referenced requirement phenomena, *ref*, are concerned with monitoring the status of the system health signal, *stat_cat*, provided by the *CS* and the combined health status of the monitored systems, *stat_sys*, provided by *SYS*. The constrained requirement phenomena, *cons*, are concerned with playing the selected warning message, *sel_audio*,

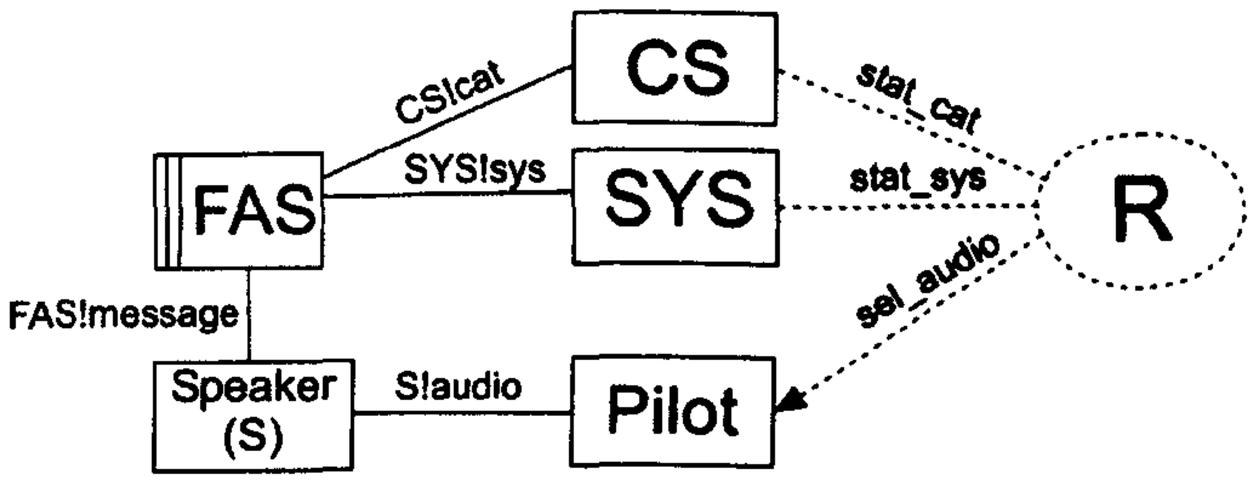

Figure F.1: The Warning System

to the pilot.

Where: $cons = \{sel\_audio\}$ and $ref = \{stat\_cat,\ stat\_sys\}$

The initial software problem is as follows:

$$P_{Initial}: \begin{array}{l} CS(stat\_cat)^{cat},\ SYS(stat\_sys)^{sys},\ Pilot(sel\_audio)_{audio}, \\ Speaker^{audio}_{message},\ FAS^{message}_{cat,sys} \end{array} \vdash R^{sel\_audio}_{stat\_cat,stat\_sys}$$

This model can be represented as a PF problem diagram as shown in Figure F.1.

The high level *FAS* requirement, *R*, is as follows:

Ra When the *CS* or a monitored system has failed, the *FAS* system shall play the correct audio message to the pilot.

Rb Message levels shall be comfortably heard by the Pilot.

Rc If more than one system has failed messages shall be selected for play in the order: *stat_cat* fail , *stat_sys* fail.

Rd If no system failures are detected, then no message shall be played.

RS For hazards H1 and H2 defined above, their respective safety targets (10^{-7}fpfh and 10^{-5}fpfh) shall be satisfied.

Note that a complete statement of R should also include requirements that cover space, weight, interfaces, maintenance and so on, but these are beyond the scope of this work.

CLAIM: *The interpretations are well-founded*

ARGUMENT & EVIDENCE: The choice of domains follows from the aircraft level safety analysis and the previous successful developments that used the *CS* and *SYS* domain structure. The pilot headphones, *Speaker*, is an existing component of the avionics system, with well-known, validated properties. The *Pilot* is trained to follow protocols rigorously. The *FAS* is to be designed, but the Company are domain experts for this type of system and have the success of a prototype to base the development on.

The customer functional requirement (**Ra** to **Rd**) was provided as an input to the developer team, and was validated with the customer.

CLAIM: *The architecture is feasible*

ARGUMENT & EVIDENCE: The selected high-level system architecture has been used successfully on previous developments.

CLAIM: *Sound judgement followed*

ARGUMENT & EVIDENCE: Development based on IPDP used successfully on many previous developments and enhanced with POSE to mitigate some known IPDP issues. Therefore it follows normal and not radical design concepts as defined in [131].

CLAIM: *Domain components are reliable*

ARGUMENT & EVIDENCE: The *CS*, *SYS* and *Speaker* domains have the required reliability, details omitted for reasons of brevity. The *Pilot* is a highly trained, competent individual.

The second POSE safety pattern activity of Solution Interpretation and Expansion involves expanding the *FAS* to include a candidate solution architecture.

The activities associated with this task are represented in the step WPS2 shown in section 6.3.2.

F.2 PPT Applied to the *Pilot* Domain

Use the POSE PPT to remove a domain adjacent to the requirement – this is the *Pilot* domain.

WPS3: Application of the PPT for Pilot Domain Removal to problem P_{Exp} to form P_{Red1}

INCLUDES: None

JUSTIFICATION J_3: The *Pilot* domain removal is based on the process given in section 5.1.2. The *Pilot* domain is adjacent to the requirement $R1$. The *Pilot* domain is constrain-type. The *Pilot* domain *effect is hear the warning message audio played through the headphones, and (implicitly) understand it.* However, although the *Pilot* is trained to follow protocol rigorously, there is a chance a mistake could be made with respect to hearing the message (albeit unlikely) and any analysis has to take due account of this – this information is stored as part of the assumption and justification for the removal. The assumption (A31) is that if the warning message audio is correctly sent to the headphones, then the Pilot will (highly likely) hear and understand the correct message.

The requirement relating to the Pilot is transformed to relate to the Speaker domain. For example, **R1a** contains the term "play the correct audio message to the Pilot". Under the transformation, this will be rephrased in **R2a** to refer to the Speaker by making (and justifying) transformations from Pilot-oriented phenomena into Speaker-oriented phenomena. In this case the re-phrase will be to "play the correct *audio* message through the *Speaker*". **R1b** will be updated in **R2b** to require that the audio

message sound levels are adequately above the ambient level (in this case 60 dB). This will add the assumption (A32) that the Pilot can actually hear the message as it issues from the Speaker, i.e. 60 dB above ambient is adequate. That we rely on the assumptions A31 and A32 holding is the justification J_3 for this progression.

Transforming $R1$ in this way yields a new requirement called $R2$, in which **R1a** becomes **R2a** and **R1b** becomes **R2b**. Requirements **R1c** and **R1d** are not changed by the transformation (they do not mention Pilot phenomena) and become **R2c** and **R2d** unchanged, respectively. **RS** also remains unchanged.

These requirement transformations do not require additional information not already in the interface nor do they refer to future events. However they do rely on information from the domain being removed being "saved" as part of the assumption that justifies the transformation. That is, we have that *Pilot*, $R2 \vdash R1$ which states that $R2$ requires assistance from the *Pilot* domain environment to satisfy the higher level requirement $R1$. Therefore this is an example of an *Environment Constraint* requirement.

R2a When the *CS* or a monitored system has failed, the *FA* shall control the system to play the correct *audio* message **through the** *Speaker*.

R2b Message levels shall be **at 60dB above the ambient cockpit level.**

R2c If more than one system has failed messages shall be selected for play in the order: *stat_cat* fail , *stat_sys* fail.

R2d If no system failures are detected, then no message shall be played.

RS For hazards H1 and H2 defined above, their respective safety targets (10^{-7}fpfh and 10^{-5}fpfh) shall be satisfied.

The domain *Pilot* is no longer referenced in the model and can be removed.

The problem is transformed from P_{Exp} into P_{Red1} which is shown in POSE sequent form as follows.

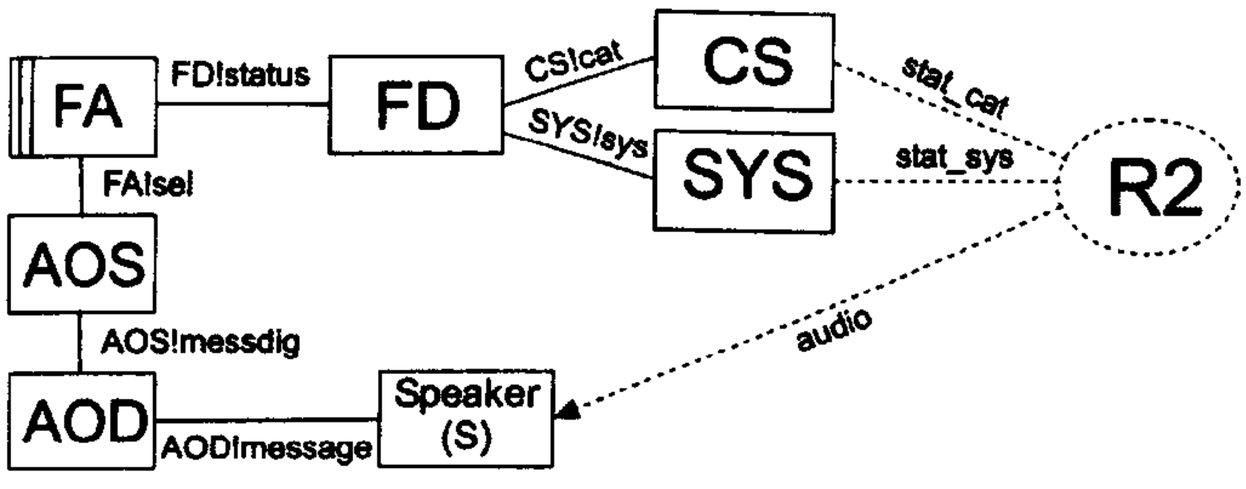

Figure F.2: The Warning System After Pilot Removal

$$P_{Red1}: \quad \begin{array}{l} CS(stat\_cat)^{cat},\ SYS(stat\_sys)^{sys},\ Speaker(audio)_{message}, \\ AOS_{sel}^{messdig},\ AOD_{messdig}^{message},\ FD_{cat,sys}^{status},\ FA_{status}^{sel} \end{array} \quad \vdash \quad R2_{stat\_cat,stat\_sys}^{audio}$$

The reduced Warning System is shown in PF problem diagram form in Figure F.2.

CLAIM: *The pilot's intention is correctly represented by the phenomena relationships*

ARGUMENT & EVIDENCE: The constraint being relied upon is that the *Pilot effect* of hearing and understanding the audio *sel_audio* corresponds directly to the phenomena *audio* generated by the *Speaker* domain. This is satisfied as noted above (e.g. refer to the derivation of **R2b**).

PHENOMENA: None

F.3 PPT Applied to the *Speaker* Domain

The next domain removed is the *Speaker*.

WPS4: Application of the PPT for Speaker Domain Removal to problem P_{Red1} to form P_{Red2}

INCLUDES: None

JUSTIFICATION J_4: The *Speaker* domain removal is based on the process given in section 5.1.2. The *Speaker* domain is adjacent to the requirement *R2*. The *Speaker* domain is constrain-type. The *Speaker* domain *effect* is *play the selected warning message audio through the headphones.* This *audio* effect is caused by the *message* audio signal output from the *AOD*.

In this case *R2* is transformed into *R3*. This time **R2a** that refers to the *Speaker* has to be re-phrased in terms of the *AOD* audio output stream (*message*) that drives the *Speaker*, to become R3a. **R2b** requires the message audio signal levels to be 60 dB above ambient, this translates into **R3b** requiring the *message* audio amplitude to be within the defined range (to achieve the 60 dB above ambient at the headphones). As before **R1c** becomes **R2c** without change. Next consider transforming **R2d** into **R3d**. The term "no message played" through the *Speaker* translates into "no analogue audio message generated" at the *AOD*. The *Speaker* is a component of the headphones

system which has a high reliability and is checked/validated by other audio functions – so no failure budget is required for the *Speaker* domain and therefore **RS** is also unchanged.

The domain removal assumption (A4) is that generating the correct audio *message* at the output of *AOD* will cause the correct *audio* message stream to correctly drive the *Speaker* within the specified comfortable audio range, to produce the desired warning message (justification J_4).

Therefore the PPT application transforms the problem from P_{Red1} into P_{Red2}, and $R2$ is transformed into $R3$ as follows:

R3a When the *CS* or a monitored system has failed, the *FA* shall control the system to **generate the correct audio *message* sequence at the output of *AOD***

R3b The AOD *message* audio amplitude levels shall be within the defined range.

R3c If more than one system has failed messages shall be selected for play in the order: *stat_cat* fail , *stat_sys* fail.

R3d If no system failures are detected, then **no analogue audio message shall be generated.**

RS For hazards H1 and H2 defined above, their respective safety targets (10^{-7}fpfh and 10^{-5}fpfh) shall be satisfied.

The constraint being relied upon is that the *Speaker effect audio* output level corresponds directly to the input phenomena *message* audio signal level generated by the *AOD* domain.

The domain *Speaker* is no longer referenced in the model and can be removed.

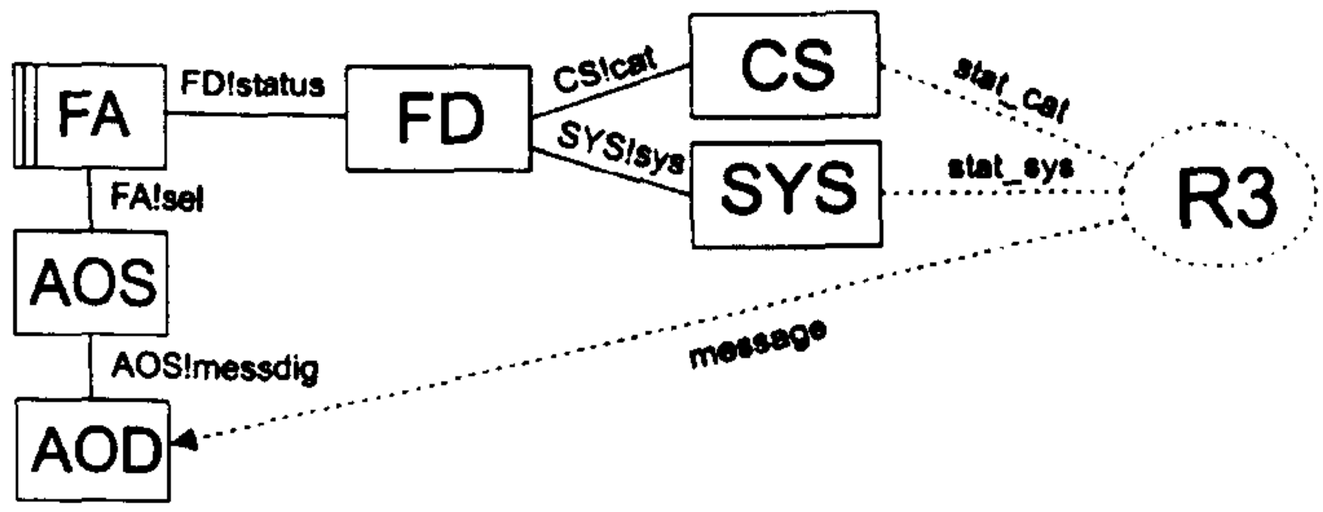

Figure F.3: The Warning System After *Speaker* Removal

The software problem is transformed into P_{Red2}, which is shown in POSE sequent form as follows.

$$P_{Red2}: \quad \begin{array}{l} CS(stat\_cat)^{cat},\ SYS(stat\_sys)^{sys},\ AOS_{sel}^{messdig}, \\ AOD(message)_{messdig},\ FD_{cat,sys}^{status},\ FA_{status}^{sel} \end{array} \quad \vdash \quad R3_{stat\_cat,stat\_sys}^{message}$$

The reduced Warning System is shown in PF problem diagram form in Figure F.3.

PHENOMENA: None

F.4 PPT Applied to the *AOD* Domain

WPS5: Application of the PPT for AOD Domain Removal to problem P_{Red2} to form P_{Red3}

INCLUDES: None

JUSTIFICATION J_5: The *AOD* domain removal is based on the process given in section 5.1.2. The *AOD* domain is adjacent to the requirement *R*3. The *AOD* domain is constrain-type. The *AOD* domain *effect* is *output correct analogue audio stream of the selected message*. This *audio* effect is caused by the *messdig* digital input stream

of the selected message from the *AOS*. The *AOD* consists of a Digital-to-Analogue Converter (DAC) to convert the digital input stream into an analogue signal equivalent, this is then passed through an amplifier to boost the signal to a level that ensures the analogue output is capable of driving the headphones at an appropriate level – there is also volume limiting circuitry to ensure that the output signal does not overdrive the headphones. The *AOD* does not affect H1, but its failure to produce an analogue output can cause H2, so some of the H2 failure budget (10^{-5}fpfh) must be allocated to it.

In this case *R*3 is transformed into *R*4. Therefore **R3a** that refers to the *AOD* and *message* needs to be re-phrased as **R4a** in terms of the input stream *messdig* and the domain *AOS*. **R3b** refers to signal levels being within a defined range. This is implicitly encoded in the operation of the *AOD* domain and has no relevance once the *AOD* has been removed. As before **R1c** becomes **R2c** without change. Next consider transforming **R3d** into **R4d**. The term "no analogue audio message shall be generated" translates into "no digital message stream shall be generated". **RS** is also unchanged apart from noting that some of the failure budget for H2 must be assigned to the *AOD*. This is captured as part of the assumption generated by the removal of *AOD*. Therefore **RS** becomes **R4S**.

The assumption (A5) associated with removing the *AOD* domain is that "the *AOD* converts the selected digital input message stream into an analogue equivalent signal which is amplified to a level to ensure that it will adequately drive the pilot's headphones. Part of the H2 failure budget must be assigned to the *AOD*".

Therefore the PPT application transforms the problem from P_{Red2} into P_{Red3}, and *R*3 is transformed into *R*4 as follows:

R4a When the *CS* or a monitored system has failed, the *FA* shall control the system to **generate the correct digital message stream sequence,** ***messdig*****, at the**

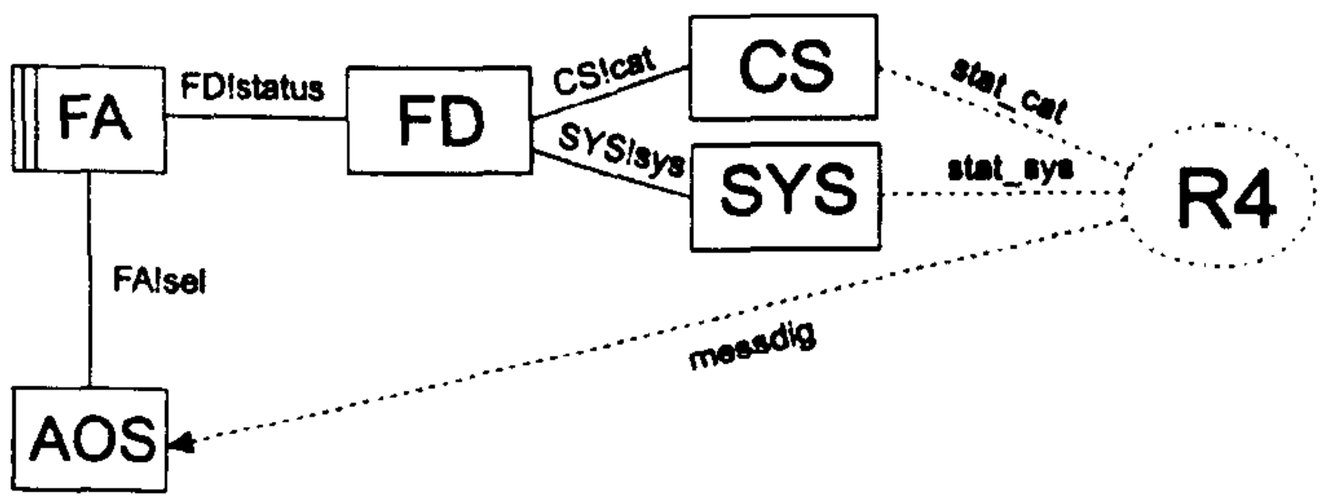

Figure F.4: The Warning System After *AOD* Removal

output of *AOS*

R4b Null

R4c If more than one system has failed messages shall be selected for play in the order: *stat_cat* fail , *stat_sys* fail.

R4d If no system failures are detected, then **no digital message stream shall be generated.**

R4S For hazards H1 and H2 defined above, their respective safety targets (10^{-7}fpfh and 10^{-5}fpfh) shall be satisfied. (Note some of H2 budget used for *AOD*).

The constraint being relied upon is that the *AOD effect message* (analogue output) signal corresponds directly to the digital input phenomena *messdig* signal generated by the *AOS* domain.

The domain *AOD* is no longer referenced in the model and can be removed.

The software problem is P_{Red3}, which is shown in POSE sequent form as follows.

$$P_{Red3}: \quad \begin{array}{ll} CS(stat\_cat)^{cat},\ SYS(stat\_sys)^{sys},\ AOS(messdig)_{sel}, & \\ FD^{status}_{cat,sys},\ FA^{sel}_{status} & \vdash\ R4^{messdig}_{stat\_cat,stat\_sys} \end{array}$$

The reduced Warning System is shown in PF problem diagram form in Figure F.4.

PHENOMENA: None

F.5 PPT Applied to the *SYS* Domain

WPS6: Application of the PPT for SYS Domain Removal to problem P_{Red3} to form P_{Red4}

INCLUDES: None

JUSTIFICATION J_6: The *SYS* domain removal is based on the process given in section 5.1.2. The *SYS* domain is adjacent to the requirement *R*4. The *SYS* domain is reference-type. The *SYS* domain *effect* is *output the health status, stat_sys, of the monitored systems.* This *stat_sys* effect causes the *sys* phenomena at the input to the *FD*.

In this case *R*4 is transformed into *R*5. The phenomena *stat_sys* is not involved in **R4a**, **R4b** or **R4d**, so these remain unchanged for *R*5. However *stat_sys* does occur in **R4c** and this is modified by replacing the *effect stat_sys* with the phenomena *sys* that it causes in **R5c**. The failure of *SYS* cannot impact on H1 or H2 which relate to the *cat* phenomena because of the selected architecture – therefore **R4S** becomes **R5S**.

The assumption (A6) associated with removing the *SYS* domain is that "the *SYS* converts the individual health statuses of the monitored systems into a combined health status value represented by *sys*".

Therefore the PPT application transforms the problem from P_{Red3} into P_{Red4}, and *R*4 is transformed into *R*5 as follows:

R5a When the *CS* or a monitored system has failed, the *FA* shall control the system

to generate the correct digital message stream sequence, *messdig*, at the output of *AOS*.

R5b Null

R5c If more than one system has failed messages shall be selected for play in the order: *stat_cat* fail , *sys* **fail.**

R5d If no system failures are detected, then no digital message stream shall be generated.

R5S For hazards H1 and H2 defined above, their respective safety targets (10^{-7}fpfh and 10^{-5}fpfh) shall be satisfied. (Note some of H2 budget used for *AOD*, but *SYS* has no impact).

The constraint being relied upon is that the *SYS effect stat_sys* corresponds directly to the phenomena *sys* that is input to the *FD* domain.

The domain *SYS* is no longer referenced in the model and can be removed.

The software problem is P_{Red4}, which is shown in POSE sequent form as follows.

$$P_{Red4}: \quad \begin{array}{l} CS(stat\_cat)^{cat},\ AOS(messdig)_{sel}, \\ FD(sys)^{status}_{cat},\ FA^{sel}_{status} \end{array} \vdash R5^{messdig}_{stat\_cat,sys}$$

The reduced Warning System is shown in PF problem diagram form in Figure F.5.

PHENOMENA: None

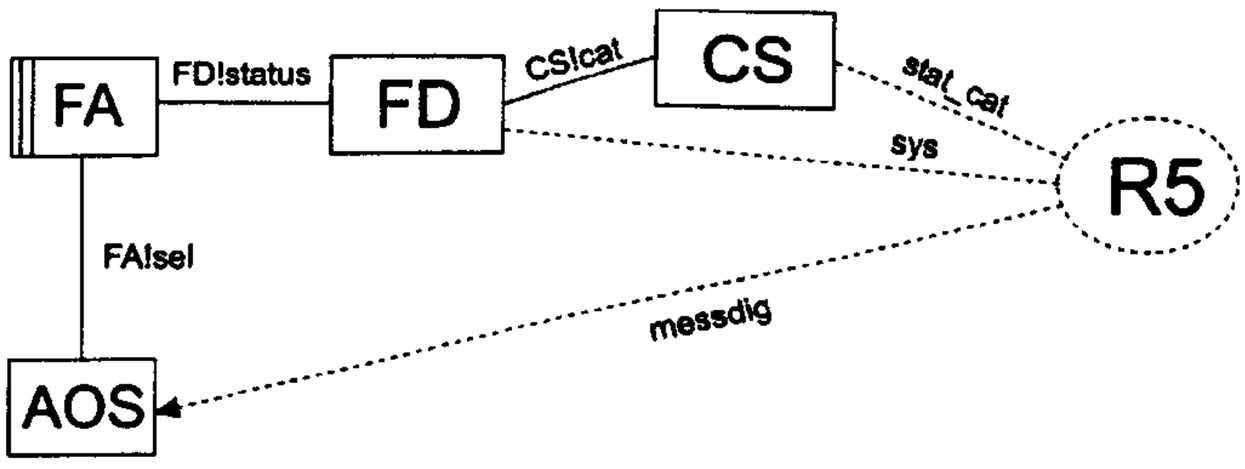

Figure F.5: The Warning System After *SYS* Removal

F.6 PPT Applied to the *CS* Domain

WPS7: Application of the PPT for CS Domain Removal to problem P_{Red4} to form P_{Red}

INCLUDES: None

JUSTIFICATION J_7: The *CS* domain removal is based on the process given in section 5.1.2. The *CS* domain is adjacent to the requirement *R*5. The *CS* domain is reference-type. The *CS* domain *effect* is *output the health status, stat_cat, of the Catastrophic System, CS*. This *stat_cat* effect causes the *cat* phenomena at the input to the *FD*.

In this case *R*5 is transformed into *R*6. The phenomena *stat_cat* is not involved in **R5a**, **R5b** or **R5d**, so these remain unchanged for *R*6. However *stat_cat* does occur in **R5c** and this is modified by replacing the *effect stat_cat* with the phenomena *cat* that it causes in **R6c**. The failure of *CS* can impact on both H1 or H2, therefore some of the failure budgets for both H1 and H2 must be allocated to this *CS* domain – **R5S** becomes **R6S**.

The assumption (A7) associated with removing the *CS* domain is that "the *CS* converts the health status of the Catastrophic System into a health status value represented by *cat*. Some of the failure budget for H1 and H2 must be allocated to *CS*".

Therefore the PPT application transforms the problem from P_{Red4} into P_{Red}, and *R*5 is transformed into *R*6 as follows:

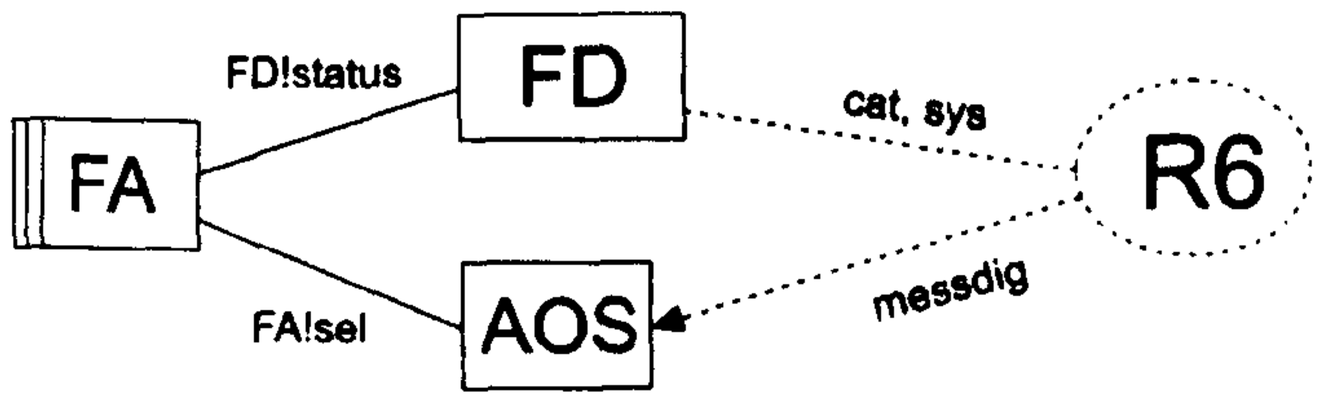

Figure F.6: The Warning System After *CS* Removal

R6a When the *CS* or a monitored system has failed, the *FA* shall control the system to generate the correct digital message stream sequence, *messdig*, at the output of *AOS*.

R6b Null

R6c If more than one system has failed messages shall be selected for play in the order: *cat* **fail** , *sys* fail.

R6d If no system failures are detected, then no digital message stream shall be generated.

R6S For hazards H1 and H2 defined above, their respective safety targets (10^{-7}fpfh and 10^{-5}fpfh) shall be satisfied. (Note some of H1 budget used for *CS*; some of H2 budget used for *AOD* and *CS*, but *SYS* has no impact).

The constraint being relied upon is that the *CS effect stat_cat* output level corresponds directly to the phenomena *cat* that is input to the *FD* domain.

The domain *CS* is no longer referenced in the model and can be removed.

The software problem is P_{Red}, which is shown in POSE sequent form as follows.

$$P_{Red}: \quad AOS(messdig)_{sel}, FD(cat, sys)^{status}, FA^{sel}_{status} \quad \vdash \quad R6^{messdig}_{cat,sys}$$

The reduced Warning System is shown in PF problem diagram form in Figure F.6.

PHENOMENA: None

F.7 POSE/Alloy Model of the Modified Audio Warning System

```
//### Problem P'Equiv: Modified AOS & Time Control  FD ###
module AudioWarn
 open util/ordering[Time] as Ti
 // open  NatNumbers as nat
 // open Derek/Alloy4/NatNumbers as nat

 // Define the Time Component
sig Time {}

// Define the OK Component
sig OK {}
one sig  yes, noo extends OK {}

// Define the Status Component
sig Status {}
 one sig  catnon, sysnon , catsys, non extends Status {} //#1

 // Define the Select Message Component
sig Sel {}
one sig  m1, m2, m3, ni extends Sel {}

// Define the Message Component
sig Mess {}
one sig  mes1, mes2, mes3, noni extends Mess {}


// Define the single FA State - sel and mute
one sig FA { sel : Sel -> Time, mute : OK -> Time } {}

// Define the single FD State domain - status
one sig FD {status : Status -> Time} {}

// Define the single AOS' State domain - messdig and id
one sig AOS {mes : Mess -> Time, id : OK -> Time} {}


// #### DEFINE OPERATIONS #####
```

```
//  FABehave models the behaviour of the safety critical controller.
pred FABehave(t,t' : Time) { FA.sel.t' = extract[FD.status.t] &&
                                          FA.mute.t' = checkid[t,t'] }

// FDBehave models the behaviour of the FD domain
pred FDBehave (t,t' : Time) { trange[t, 0, 1] => FDSame[t,t']
     else trange[t, 2, 4]  => (FD.status.t' = catnon )
          else trange[t, 5, 6] => (FD.status.t' = non )
               else trange[t,7,9] => (FD.status.t' = sysnon )
                    else trange[t,10,11] =>  (FD.status.t' = non )
                         else trange[t,12,13] =>  (FD.status.t' = catsys )
                              else trange[t,14,15] =>  (FD.status.t' = non )
                                   else trange[t,16,24] =>  (FD.status.t' = catnon )
                                        else trange[t, 25, 26] => (FD.status.t' = non )
                                             else FDSame[t,t'] }

// AOSBehave models the behaviour of the AOS' domain. If mute is
// active then the message is set to noni (no message).
pred AOSBehave  (t,t' : Time) { AOS.mes.t' = (FA.mute.t = noo =>
                                                decode[FA.sel.t] else noni) &&
                                             AOS.id.t' = setid[t,t']}

pred FDSame (t,t' : Time)     { FD.status.t' = FD.status.t }


// Predicate trange returns true if t is in the range from trs to tf.
pred trange [t: Time, ts,tf: Int] {#prevs[t] >= ts &&    //#6
                                   #prevs[t] =< tf}

// Fun extract returns the message selected by the FD status input (status)
fun extract[st: Status] : Sel {(st=catnon or st=catsys)  //#7
                  => m1 else  st = sysnon => m2 else ni}

// Fun decode returns the message selected by the FA status input (sel)
fun decode[s: Sel] : Mess { s = m1 => mes1                    //#8
                  else  s = m2 => mes2 else noni}
```

```
// Fun checkid confirms message id corresponds to selected message #11
fun checkid[t,t' : Time]: OK {AOS.id.t = yes => noo else yes}


// Fun setid determines if message identifier, id, in AOS' is ok at specified time  #12
fun setid[t,t' : Time]: OK { trange[t, 0, 18] => yes
                             else  trange[t, 19, 23] => noo
else yes }

//#### Define Trace Model #####                          #9

pred init [t : Time] {FD.status.t  = non && FA.sel.t = ni && AOS.mes.t = noni
                                 && FA.mute.t = yes && AOS.id.t = noo }


fact traces {
   init[Ti/first[]]
   all t  : Time - Ti/last[] | let t' = Ti/next[t] |  FDBehave[t,t'] && FABehave[t,t']  &&
                                                      AOSBehave[t,t'] }

// ##### PROPERTIES #####                  #10

pred show1() {}
run show1  for   30  but 6 int  //Set bit width to cover -32 to +31
```

Appendix G

Some Further Feasible Requirement Examples

This chapter considers the other two examples of non-feasible/feasible requirement as discussed in section 2.4.

G.1 A Future Reference Example

Future Reference covers the case when the satisfaction of the overall system requirement (R) requires information that is available only in the future. Its solution often involves either (a) sharing additional information (similar to Unshared Information) or (b) making reasonable assumptions about the required behaviour. In both cases use is made of domain information to provide a solution.

In the Audio Warning *FAS* messages are played in a hierarchy (e.g. **R7c**) with *cat* fail messages having the highest priority. Therefore, if a *sys* message is being played, and a *cat* message is detected, then the required behaviour is that the *sys* warning message should be stopped, and the *cat* warning message started as quickly as possible. If a *sys* message is being played and another *sys* failure is reported from a different monitored system, the question arises as to when should this new

message be played? The desired behaviour is that it should be played after the current message has finished. However, how does the system know when a message has finished? This is a *Future Reference* issue and there are two possible design solutions based on the solution types discussed above, namely: (a) based on sharing additional information and (b) based on making reasonable assumptions about a known property of the system.

The first alternative solution requires the derived requirement $DR-A$ to be added to the requirement associated with the software problem $P_{Equiv'}$. $DR-A$ has the form:

> Include a flag mechanism for denoting the end of a message, which is flagged to the *FA* from the AOS' when detected, so that the *FA* knows the currently playing message is completing and the next can be started.

In addition the phenomena *eom* (End Of Message) needs to be added to allow the AOS' to flag to the *FA* that the current message has finished. These behaviour refinements are added to the problem domain through a series of POSE transformations. These are included into the problem domain through a step, WPS8A, which includes the appropriate justifications. The step will include the following transformations (with associated justifications): (a) requirement interpretation to include $DR-A$, (b) domain interpretation to introduce the phenomena *eom* into AOS', (c) solution interpretation to include a reference to the phenomena *eom* into *FA*, and (d) the expansion transformations that include these interpretation transformations into the software problem which transforms from P_{Equiv} into P_{solA}.

The second alternative solution requires the derived requirement $DR-B$ to be added to the requirement associated with the software problem $P_{Equiv'}$. $DR-B$ has the form:

> All messages last less than five seconds in duration and the *FA* knows when a message is started. Use this information to start playing the next

message, five seconds after the current message was started and so on.

No new interface phenomena are need to be added to the problem, since *FA* knows when it requested the start of a message and it can run a clock to count off the 5 second duration of the message. Therefore the step, WPS8B, associated with this alternative solution only requires a requirement interpretation/expansion transformation pair to be justified.

Comparison of the two alternatives indicates that the second approach is the easiest to implement and so it was adopted in this case. The main factor favouring the second alternative was that there was no need to complicate the interface between the *FA* and the AOS' with new phenomena.

G.2 Directly Implementable Requirement : Emergency Stop function

The fourth type of requirement discussed in section 2.4 are those that are directly implementable. A typical example is shown in Figure G.1 which satisfies a requirement for an emergency stop (*ES*) that is added to the list of the Audio Warning, *FAS*, requirement with the aim of mitigating the "Message played too loud" issue identified in section 6.3.6. The idea being that the Pilot uses the emergency stop button to terminate warning messages from the *FAS* that are uncomfortably loud.

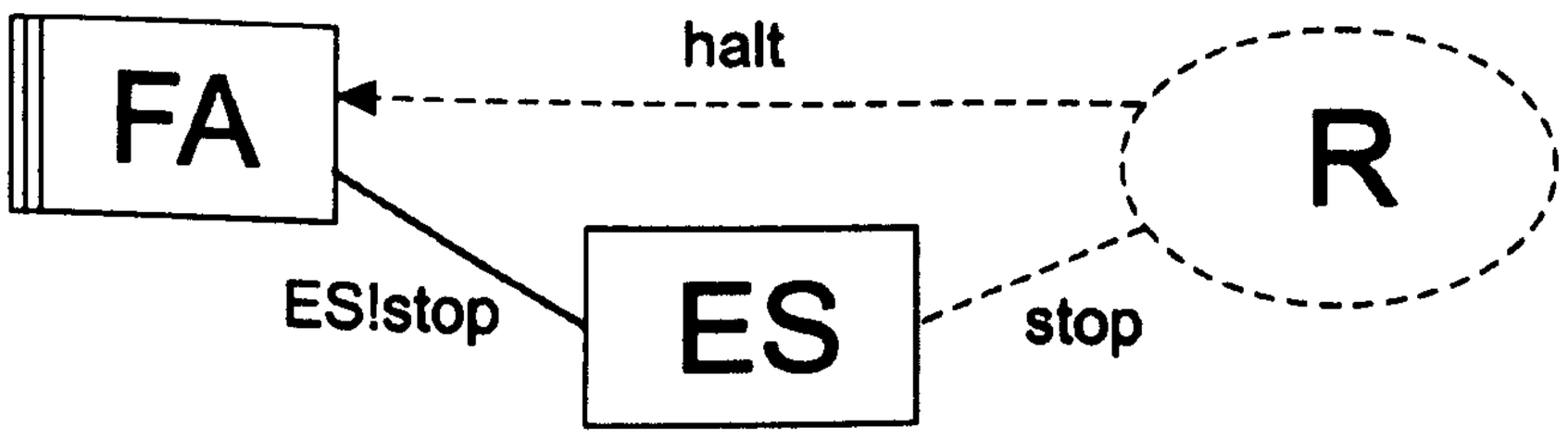

Figure G.1: The Emergency Stop Function

The requirement is that message processing in the *FA* should halt immediately if the emergency stop button is pressed. In Figure G.1 the emergency stop button

is represented by the *ES* domain which shares the stop phenomena with the *FA*. When the *ES* button is pressed, the stop input to the *FA* causes the *FA* to halt processing as required. The diagram shows that the requirement phenomena (*halt*, ***stop***) relate directly to the *FA* phenomena relevant to the emergency stop processing. There are no intervening domains that need to be removed or requirement phenomena that have to be transformed. Therefore a directly implementable requirement can be identified from the PF diagram by (a) there is at most a single functional domain between the to be designed domain (*FA* in this case) and the requirement, and (b) the requirement phenomena are equivalent to the design phenomena. In situations where the requirement phenomena and design phenomena are close, but do not quite coincide, then the application of *Sharing Removal* from the PPT can solve the problem. This is similar to its use to transform P_{Red} into P_{Equiv} at the end of section 6.3.3. Also, the term "functional domain" is used to preclude the introduction of dummy identity domains which in theory could introduce an arbitrary number of domains between the to be designed domain and the requirement without affecting functionality.

Appendix H

Revised *DC* Case Study

The aim of this case study is to show that POSE works well with requirements presented in a standardised good practice form based on [118], and to again show that utility of applying the full POSE safety pattern based on the work in Chapter 6.

H.1 POSE Safety Pattern Activity 1

The goal is to derive a machine specification that can satisfy its safety requirements (**H1** and **H2**) and thus form a good basis for the rest of the development process, and this is achieved by applying the POSE safety pattern introduced in Chapter 6. The first activity is *Context and Requirement Interpretation* and this uses the requirements and domain information based on Appendix A.3, except this time the aim is to use a structured form of the requirements based on current Company procedures [118] and the **abstraction heuristic check** (i.e. a requirement should only refer to domains and phenomena that are adjacent to it). The resulting set of requirements R are:

R.1 The DS shall command (DS_sel) which flare is selected for release from the DU (flare_select).

R.2 The DS shall command (DS_fire) when a flare is to be released by the DU (flare_fire).

R.3 The selected flare shall only be released from the DU if the DS has issued a flare release command when the input interlocks are satisfied.

R.4 The three input interlocks are:

R.4.1 The aircraft shall be in the air (*air_gnd*).

R.4.2 The safety pin shall be removed (*pin_st*).

R.4.3 The Pilot has issued an allow release command (*p_int*).

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

These requirements were reviewed and found to be apposite. The result is the Initial Problem transformation step, PSA1.

PSA1: ***Initial Problem*** *application to transform problem* P_{null} *into* $P_{Initial}$

JUSTIFICATION J_1: The identified requirement, domains and their relevant properties are summarised above. The combined justification for the set of transformations used to introduce the context, requirements and initial solution system architecture are: $J_1 = J_a \wedge J_b \wedge J_c$.

The resulting software problem is $P_{Initial}$ as follows:

$$
\begin{aligned}
& DS(DS\_sel, DS\_fire)^{cont},\ DU(flare\_select, flare\_fire)_{fire,sel}, \\
P_{Initial}: \quad & SP(pin\_st)^{pin},\ AS(air\_gnd)^{st}, \\
& P(p\_int)^{allow},\ DC^{sel,fire}_{st,pin,allow,cont} \quad \vdash \quad R'^{flare\_select,flare\_fire}_{DS\_sel,DS\_fire,air\_gnd,pin\_st,p\_int}
\end{aligned}
$$

PHENOMENA: Phenomena and their control and sharing (see $P_{Initial}$) are known from the existing system components.

CLAIM: *The interpretations are well-founded*

ARGUMENT & EVIDENCE: The choice of domains follows from the aircraft level safety analysis and the required choice of interlocks. The *DS*, *DU*, *AS* and *SP* are existing components of the avionics system, with well-known, validated properties. The ***Pilot*** is trained to follow protocols rigorously.

The customer functional requirements (**R.1** to **R.4**) were provided as an input to the developer team, and were validated with the customer. The development of the safety requirement **RS** is as discussed in section 4.1.1.

CLAIM: *The overall architecture is feasible*

ARGUMENT & EVIDENCE: The selected overall system architecture has been used successfully on similar safety systems. Inspection confirms that the identified domains and their associated phenomena can interact as needed - aim being to avoid Unshared Information issues later in the development.

CLAIM: *Sound judgement followed*

ARGUMENT & EVIDENCE: The development is based on IPDP used successfully on many previous developments and enhanced with POSE to mitigate some known issues with IPDP. Claim this follows normal and not radical design concepts as defined in [131].

CLAIM: *In-air indicator in Aircraft Status System is reliable*

ARGUMENT & EVIDENCE: The in-air indicator is obtained from the weight on wheels and landing gear up indications: if the landing gear is up and there is no weight on the wheels then the aircraft is assumed to be in the air. The landing gear is detected as being up by a number of sensor switches. The switches use a multi-pole arrangement

of appropriately selected "Normally open/Normally closed" contacts. This imbues an error detection capability that is used to achieve very good failure rates, well within the required margins.

The corresponding PF diagram form is shown in Figure H.1.

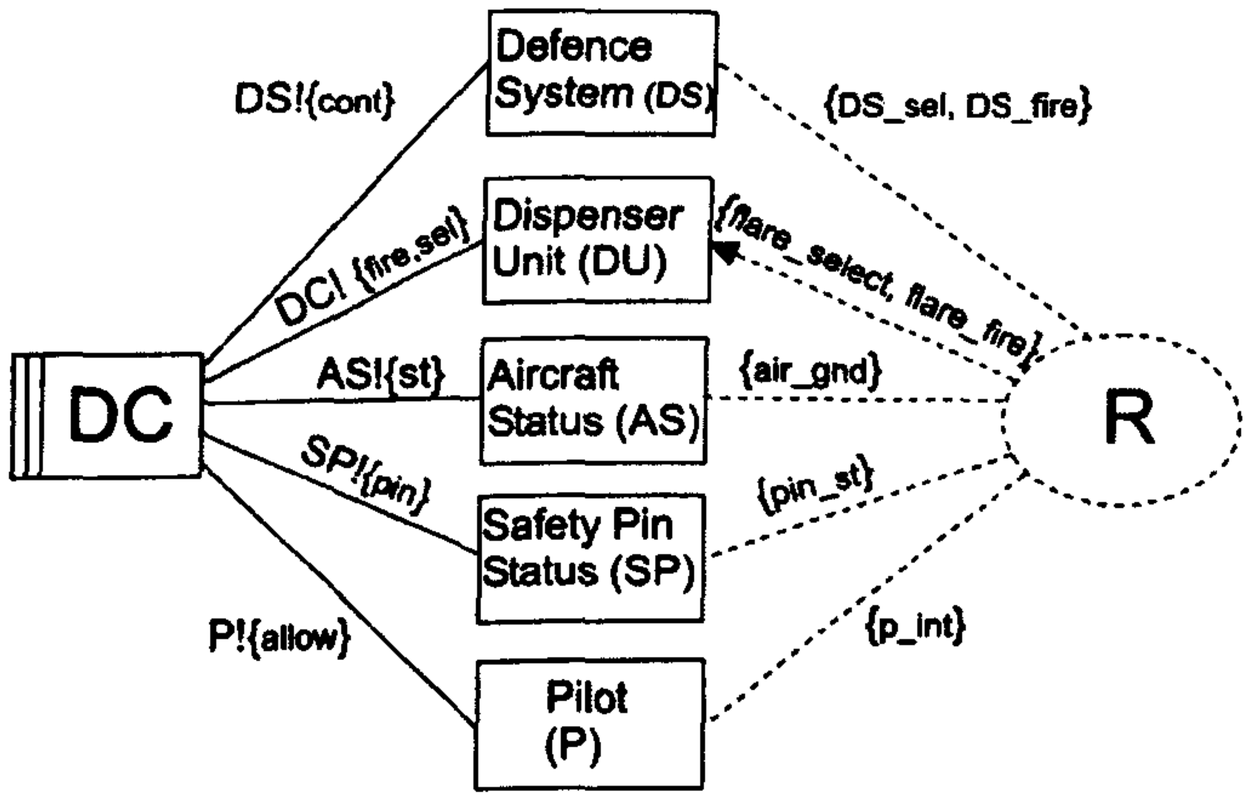

Figure H.1: PF Problem Diagram for the *DC* Problem, $P_{Initial}$

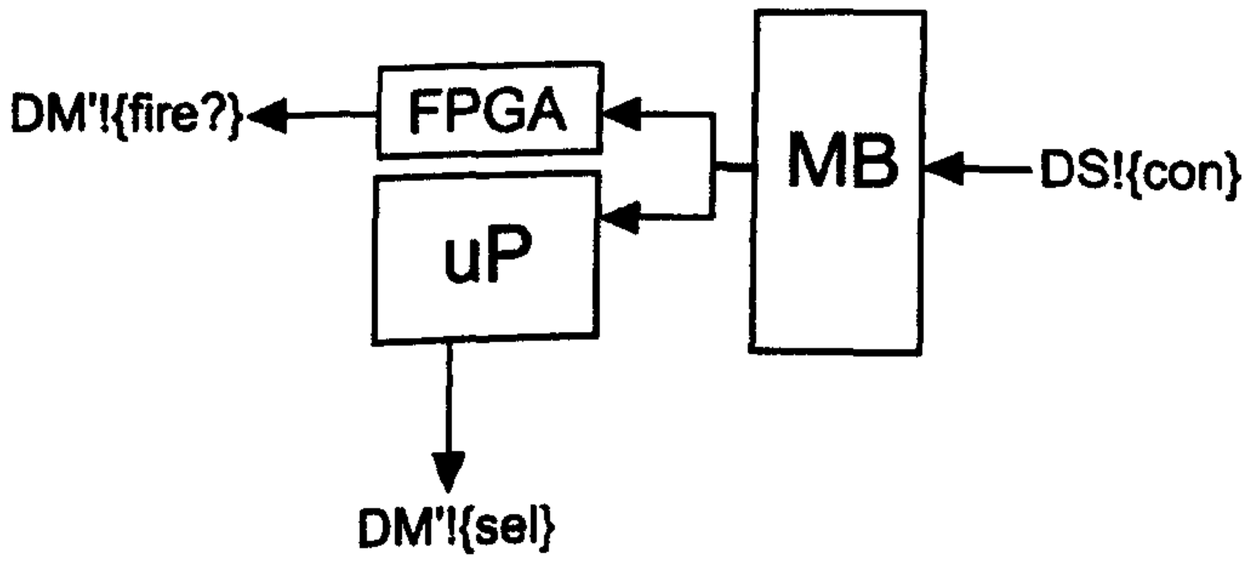

Figure H.2: Modified *DM'* Architecture

H.2 POSE Safety Pattern Activity 2

The second activity in the POSE safety pattern is Solution Interpretation and Expansion. In this case the results of the previous analysis work will be assumed and the expanded system architecture to replace the **DC** domain will be the domains **SC**, **II** and **DM** and where the **DM** domain has the modified form shown in Figure H.2. The result of this activity is the problem transformation step PSA2 shown below.

PSA2: ***Solution Interpretation and Expansion*** *applied to problem* $P_{Initial}$ *to form* P_{Exp}

JUSTIFICATION J_2: The identified system architecture, its components and relevant properties are as summarised in Figure H.3. The justification for this (architecture) solution expansion transformation is that this system architecture was successfully developed as a prototype and is known to be capable of satisfying the functional requirements.

The requirements are left unchanged by this expansion because the changes occur to domains that are not adjacent to the requirements.

The software problem is transformed from $P_{Initial}$ to P_{Exp} as shown below, and an equivalent PF representation is shown in Figure H.3.

$$
\begin{aligned}
& DS(DS\_sel, DS\_fire)^{cont},\ DU(flare\_select, flare\_fire)_{fire,sel}, \\
P_{Exp}: \quad & SP(pin\_st)^{pin},\ AS(air\_gnd)^{st},\ P(p\_int)^{allow}, \\
& II^{int}_{st,pin,allow},\ DM^{sel,fire?}_{cont},\ SC^{fire}_{int,fire?} \quad \vdash\ R^{flare\_select,flare\_fire}_{DS\_sel,DS\_fire,air\_gnd,pin\_st,p\_int}
\end{aligned}
$$

The solution system architecture is introduced using the Solution Interpretation transformation and the AStruct[...].

CLAIM: $P_{Initial}$ *and* P_{Exp} *are equivalent*

ARGUMENT & EVIDENCE: The requirement R and the interface phenomena adjacent to R are not changed by the transformation, although new phenomena are introduced to represent the interactions between the introduced domains .

CLAIM: *The choice of candidate solution architecture exhibits sound safety engineering judgement*

ARGUMENT & EVIDENCE: The system architecture is chosen to minimise the number and extent of the safety related functions, localising them to simple, distinct blocks in accordance with best practice.

CLAIM: *The chosen solution architecture does not prevent the satisfaction of R' (Feasibility).* **This claim is not yet supported.**

PHENOMENA: The new phenomena introduced by the architecture are:

fire? – Command to release the selected flare type

int – Status of combined interlocks

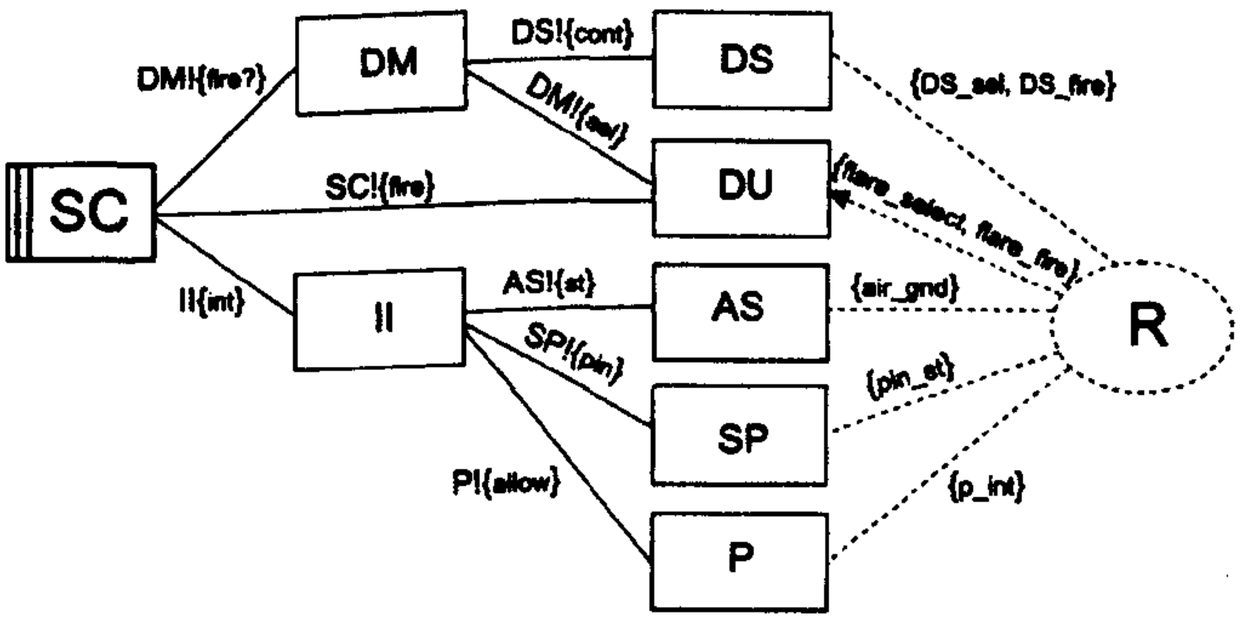

Figure H.3: Problem P_{Exp}
after Solution Interpretation and Expansion

H.3 POSE Safety Pattern Activity 3

The third activity in the POSE safety pattern is problem progression and involves applying the PPT introduced in section 5.1.2 to remove domains from the problem to

allow the machine specification to be recovered. The first domain to be transformed is the pilot domain P and the activities involved are captured in the PSA3 problem transformation step.

H.3.1 PSA3 *Pilot* Domain Removal

PSA3: Application of the PPT for Pilot Domain Removal to problem P_{Exp} to form P_{Red1}

INCLUDES: None

JUSTIFICATION J: The *Pilot* domain removal is based on the process given in section 5.1.2. The pilot P domain is adjacent to the requirement R. The P domain control phenomena *allow* maps directly to its output behaviour phenomena which is *p_int*. Therefore *p_int* (representing the pilot's intention) appears in the reference part of the requirement phenomena before the transformation, but will be replaced with its effect, *allow*, by the transformation. The requirement R is as detailed at the beginning of this section, and inspecting it indicates that only **R.4.3** involves a reference to *p_int*.

Consider the transformation from **R.4.3** into **R1.4.3** which involves the phenomena *p_int* and *allow* as noted above.

R.4.3 : The Pilot has issued an allow release command (*p_int*).

R1.4.3: *II* observes pilot input of *allow*.

Pilot Domain Description: The pilot sets the Allow switch to active to indicate that the intention is to allow flare release, otherwise it is set inactive. The Allow switch uses a multi-pole arrangement of appropriately selected "Normally open/Normally closed" contacts. This imbues an error detection capability that is used to achieve very good failure rates, well within the required margins. The setting of the Allow switch is determined by the pilot P domain and is output using the phenomena *allow*, where *allow* = yes means the Allow switch is active, whilst *allow* = no means the allow switch is inactive. Therefore *allow* = yes corresponds to allow flare release, whilst *allow* = no

corresponds to inhibit flare release.

The justification for this problem progression is that:

$R1.4.3 \wedge Assumption(p\_int, allow) \Rightarrow R.4.3.$

The $A_1 = Assumption(p\_int, allow)$ is given by the *Pilot Domain Description.* Inspection of $R1.4.3$ in conjunction with the *Pilot Domain Description* shows they imply $R.4.3$ as required.

This requirement transformation does not require additional information not already in the interface nor does it refer to future events. However it does rely on the useful behaviour from the domain being removed being "saved" as part of the assumption that justifies the transformation – it is an example of an Environment Constraint non-feasible requirement which is discussed in a POSE context in [83]. As noted above, this useful behaviour contribution is contained in the *Pilot Domain Description.*

The PPT application transforms the problem from P_{Exp} into P_{Red1}, and R is transformed into $R1$ (using the Requirements Interpretation part of the PPT) as follows:

R1.1 The DS shall command (DS_sel) which flare is selected for release from the DU (flare_select).

R1.2 The DS shall command (DS_fire) when a flare is to be released by the DU (flare_fire).

R1.3 The selected flare shall only be released from the DU if the DS has issued a flare release command when the input interlocks are satisfied.

R1.4 The three input interlocks are:

R1.4.1 The aircraft shall be in the air (*air_gnd*).

R1.4.2 The safety pin shall be removed (*pin_st*).

R1.4.3 *II* observes pilot input of *allow*.

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

The pilot domain P is no longer referenced in the model and can be removed.

The software problem is transformed to P_{Red1}:

$$P_{Red1}: \begin{array}{l} DS(DS\_sel, DS\_fire)^{cont},\ DU(flare\_select, flare\_fire)_{fire,sel}, \\ SP(pin\_st)^{pin},\ AS(air\_gnd)^{st}, \\ II(allow)^{int}_{st,pin},\ DM^{sel,fire?}_{cont},\ SC^{fire}_{int,fire?} \end{array} \vdash R1^{flare\_select,flare\_fire}_{DS\_sel,DS\_fire,air\_gnd,pin\_st,allow}$$

An equivalent PF representation is shown in Figure H.4.

CLAIM: *The pilot's intention with respect to allowing flare release is represented by the phenomena allow.*

ARGUMENT & EVIDENCE: This is covered by the *Pilot Domain Description.*

PHENOMENA: None

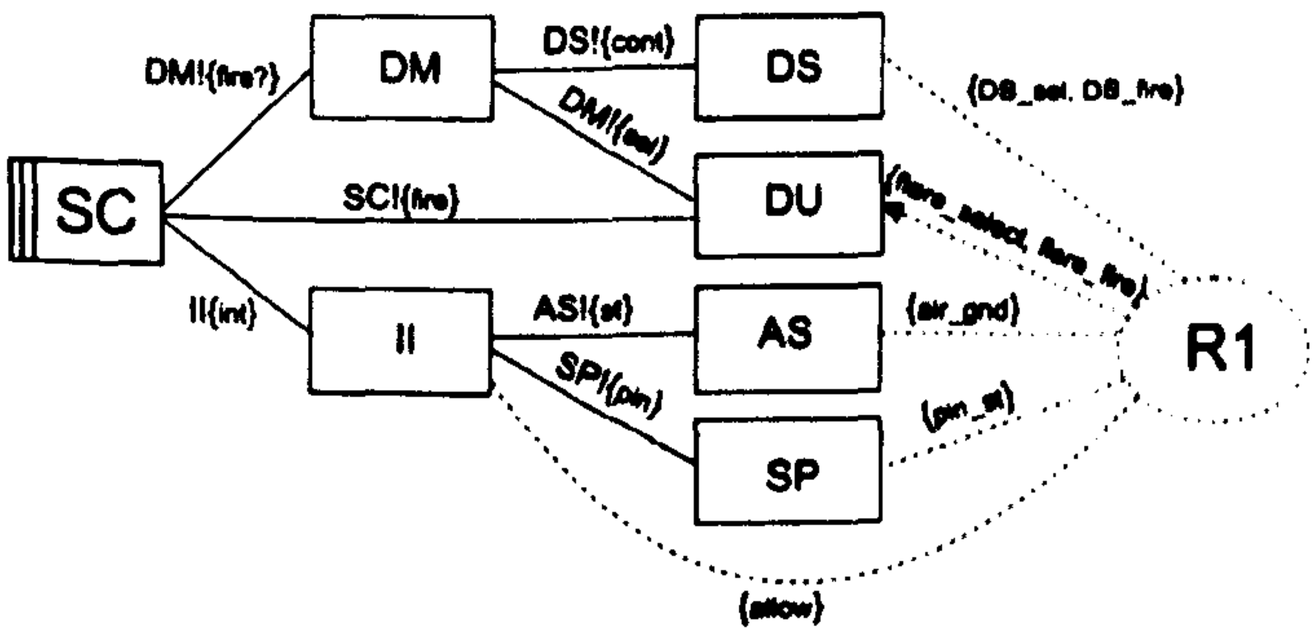

Figure H.4: Problem P_{Red1}
after *Pilot* Domain removal

H.3.2 PSA4 *SP* Domain Removal

The next domain to be removed is the safety pin domain *SP*, and the activities involved are captured in the PSA4 problem transformation step shown below.

PSA4: Application of the PPT for SP Domain Removal to problem P_{Red1} to form P_{Red2}

INCLUDES: None

JUSTIFICATION *J*: The *SP* domain removal is based on the process given in section 5.1.2. The *SP* domain is adjacent to the requirement *R*1. The *SP* domain control phenomena *pin_st* maps directly to its output behaviour phenomena *pin*. Therefore *pin_st* will appear in the reference part of the requirement phenomena before the transformation, but will be replaced with the effect it causes,*pin*, after the transformation. The requirement *R*1 is as detailed in step PSA3 on page 294. Inspection of *R*1 indicates that only **R1.4.2** involves a reference to *pin_st*.

Consider the transformation from **R1.4.2** into **R2.4.2** which involves the phenomena *pin_st* and *pin* as noted above.

*R*1.4.2 : The safety pin shall be removed (*pin_st*).

*R*2.4.2 : *II* observes input *pin* = out.

SP Domain Description : The safety pin status is obtained from sensor switches. The switches use a multi-pole arrangement of appropriately selected "Normally open/Normally closed" contacts. This imbues an error detection capability that is used to achieve very good failure rates, well within the required margins. The setting of the safety pin status is determined by the *SP* domain and is output using the phenomena *pin*, where *pin* = out means the safety pin has been removed, whilst *pin* = in means the safety pin is present (and inhibiting flare release).

The justification for this problem progression is that:

$R2.4.2 \wedge Assumption(pin\_st, pin) \Rightarrow R1.4.2.$

The $A_2 = Assumption(pin\_st, pin)$ is given by the *SP Domain Description*. Inspection of $R2.4.2$ in conjunction with the *SP Domain Description* shows they imply $R1.4.2$ as required. The constraint being relied upon is that *pin* = out means the safety pin has been removed – see claim below.

This requirement transformation does not require additional information not already in the interface nor does it refer to future events. However it does rely on the useful behaviour from the domain being removed being "saved" as part of the assumption that justifies the transformation – it is an example of an Environment Constraint non-feasible requirement which is discussed in a POSE context in [83]. As noted above, this useful behaviour contribution is contained in the *SP Domain Description*.

The PPT application transforms the problem from P_{Red1} into P_{Red2}, and $R1$ is transformed into $R2$ (using the Requirements Interpretation part of the PPT) as follows:

R2.1 The DS shall command (DS_sel) which flare is selected for release from the DU (flare_select).

R2.2 The DS shall command (DS_fire) when a flare is to be released by the DU (flare_fire).

R2.3 The selected flare shall only be released from the DU if the DS has issued a flare release command when the input interlocks are satisfied.

R2.4 The three input interlocks are:

R2.4.1 The aircraft shall be in the air (*air_gnd*).

R2.4.2 *II* observes input *pin* = out.

R2.4.3 *II* observes pilot input of *allow*.

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

The domain *SP* is no longer referenced in the model and can be removed.

The software problem is transformed to P_{Red2}:

$$
\begin{aligned}
P_{Red2}: \quad & DS(DS\_sel, DS\_fire)^{cont},\ DU(flare\_select, flare\_fire)_{fire,sel}, \\
& AS(air\_gnd)^{st},\ II(allow, pin)^{int}_{st}, \\
& DM^{sel,fire?}_{cont},\ SC^{fire}_{int,fire?} \quad \vdash\ R2^{flare\_select,flare\_fire}_{DS\_sel,DS\_fire,air\_gnd,pin,allow}
\end{aligned}
$$

An equivalent PF representation is shown in Figure H.5.

CLAIM: *The safety pin status is represented by the phenomena pin.*

ARGUMENT & EVIDENCE: This is covered by the *SP Domain Description.*

PHENOMENA: None

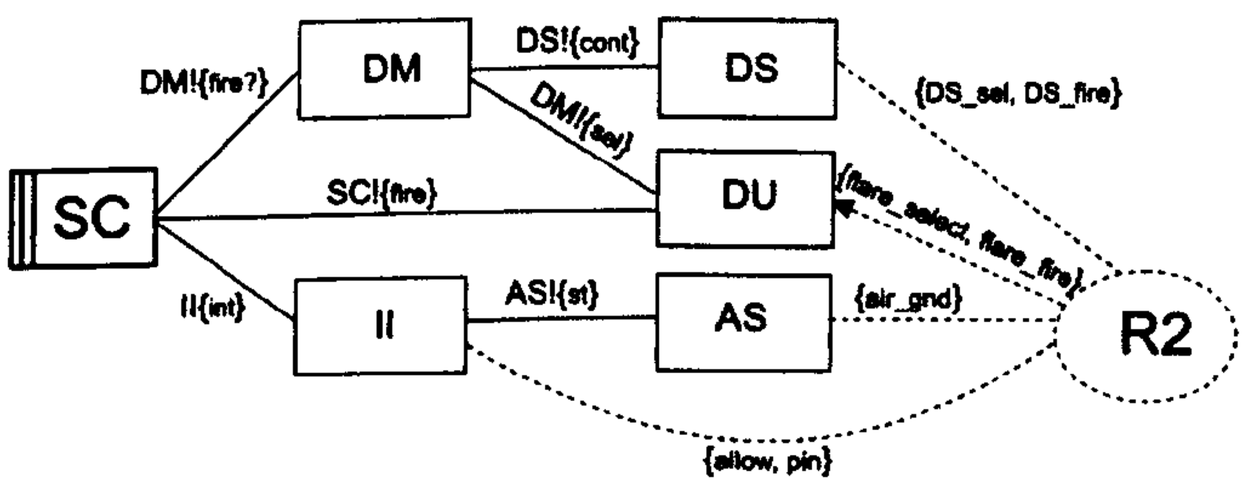

Figure H.5: Problem P_{Red2}
after *SP* Domain removal

H.3.3 PSA5 AS Domain Removal

The next domain to be removed is the air status domain AS, and the activities involved are captured in the PSA5 problem transformation step shown below.

PSA5: Application of the PPT for AS Domain Removal to problem P_{Red2} to form P_{Red3}

INCLUDES: None

JUSTIFICATION J: The AS domain removal is based on the process given in section 5.1.2. The AS domain is adjacent to the requirement $R2$. The AS domain control phenomena *air_gnd* maps directly to its output behaviour phenomena *st*. Therefore *air_gnd* will appear in the reference part of the requirement phenomena before the transformation, but will be replaced with the effect it causes,*st*, after the transformation. The requirement $R2$ is as detailed in step PSA4 on page 297. Inspection of $R2$ indicates that only **R2.4.1** involves a reference to *air_gnd*.

Consider the transformation from **R2.4.1** into **R3.4.1** which involves the phenomena *air_gnd* and *st* as noted above.

$R2.4.1$: The aircraft shall be in the air (*air_gnd*).

$R3.4.1$: II observes input st = air.

AS Domain Description : The in-air indicator is obtained from the weight on wheels and landing gear up indications: if the landing gear is up and there is no weight on the wheels then the aircraft is assumed to be in the air. The landing gear is detected as being up by a number of sensor switches. The switches use a multi-pole arrangement of appropriately selected "Normally open/Normally closed" contacts. This imbues an error detection capability that is used to achieve very good failure rates, well within the required margins. The setting of the in-air indicator is determined by the AS domain and is output using the phenomena st, where st = air means the aircraft is in the air, whilst st = gnd means the aircraft is on the ground.

The justification for this problem progression is that:

$R3.4.1 \wedge Assumption(air\_gnd, st) \Rightarrow R2.4.1.$

The $A_3 = Assumption(air\_gnd, st)$ is given by the *AS Domain Description.* Inspection of $R3.4.1$ in conjunction with the *AS Domain Description* shows they imply $R2.4.1$ as required. The constraint being relied upon is that st = air means the aircraft is in the air – see claim below.

This requirement transformation does not require additional information not already in the interface nor does it refer to future events. However it does rely on the useful behaviour from the domain being removed being "saved" as part of the assumption that justifies the transformation – it is an example of an Environment Constraint non-feasible requirement which is discussed in a POSE context in [83]. As noted above, this useful behaviour contribution is contained in the *AS Domain Description.*

The PPT application transforms the problem from P_{Red2} into P_{Red3}, and $R2$ is transformed into $R3$ (using the Requirements Interpretation part of the PPT) as follows:

R3.1 The DS shall command (DS_sel) which flare is selected for release from the DU (flare_select).

R3.2 The DS shall command (DS_fire) when a flare is to be released by the DU (flare_fire).

R3.3 The selected flare shall only be released from the DU if the DS has issued a flare release command when the input interlocks are satisfied.

R3.4 The three input interlocks are:

R3.4.1 *II* observes input st = air.

R3.4.2 *II* observes input *pin* = out.

R3.4.3 *II* observes pilot input of *allow*.

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

The domain AS is no longer referenced in the model and can be removed.

The software problem is transformed to P_{Red3}:

$$P_{Red3}: \begin{array}{l} DS(DS\_sel, DS\_fire)^{cont},\ DU(flare\_select, flare\_fire)_{fire,sel}, \\ II(allow, pin, st)^{int}, \\ DM_{cont}^{sel,fire?},\ SC_{int,fire?}^{fire} \quad \vdash\ R3_{DS\_sel,DS\_fire,st,pin,allow}^{flare\_select,flare\_fire} \end{array}$$

An equivalent PF representation is shown in Figure H.6.

CLAIM: *The in-air status of the aircraft is represented by the phenomena st.*

ARGUMENT & EVIDENCE: This is covered by the *AS Domain Description.*

PHENOMENA: None

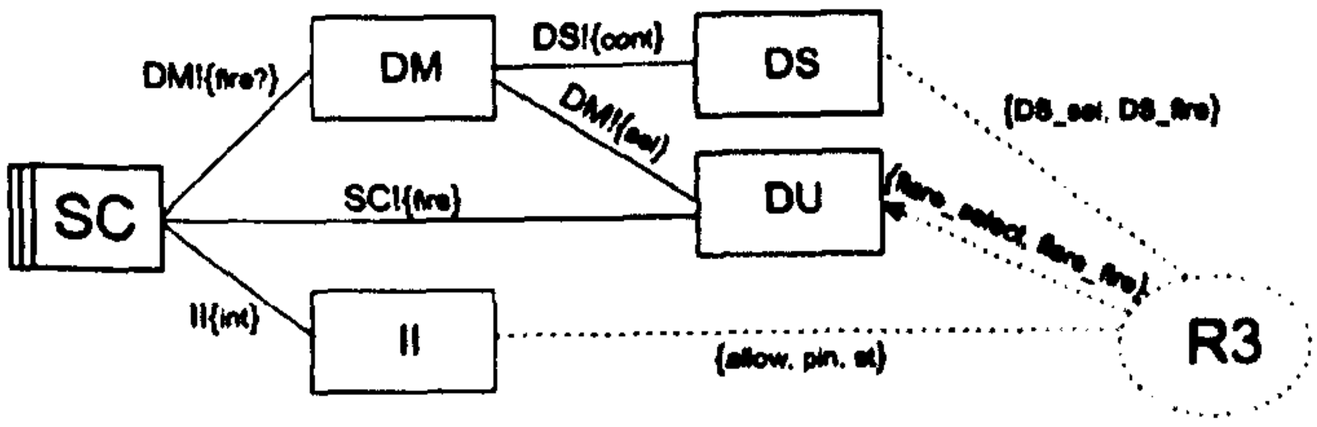

Figure H.6: Problem P_{Red3}
after AS Domain removal

H.3.4 PSA6 *DS* Domain Removal

The next domain to be removed is the Defence System domain *DS*, and the activities involved are captured in the PSA6 problem transformation step shown below.

PSA6: Application of the PPT for DS Domain Removal to problem P_{Red3} to form P_{Red4}

INCLUDES: None

JUSTIFICATION *J*: The *DS* domain removal is based on the process given in section 5.1.2. The *DS* domain is adjacent to the requirement *R*3. The *DS* domain control phenomena *DS_sel* and *DS_fire* map directly to its output behaviour phenomena *cont*. Therefore *DS_sel* and *DS_fire* will appear in the reference part of the requirement phenomena before the transformation, but will be replaced with the effect they cause,*cont*, after the transformation. The requirement *R*3 is as detailed in step PSA5 on page 300.

Inspection of *R*3 indicates that **R3.1** refers to *DS_sel*, **R3.2** involves a reference to *DS_fire* and **R3.3** refers to the *DS*.

Consider the transformation from **R3.1** into **R4.1** which involves the phenomena *DS_sel* and *cont* as noted above.

*R*3.1 : The DS shall command (DS_sel) which flare is selected for release from the DU (flare_select).

*R*4.1 : The *DM* shall decide which flare to select by observing a field in the *cont* message it receives.

Next consider the transformation from **R3.2** into **R4.2** which involves the phenomena *DS_fire* and *cont* as noted above.

*R*3.2 : The DS shall command (DS_fire) when a flare is to be released by the DU (flare_fire).

*R*4.2 : The *DM* shall decide to release a flare by observing a field in the *cont* message it receives.

Next consider the transformation from **R3.3** into **R4.3** which involves *DS* as noted above.

*R*3.3 : The selected flare shall only be released from the DU if the DS has issued a flare release command and the input interlocks are satisfied.

*R*4.3 : The selected flare shall only be released from the DU if the *DM* has received a flare release command and the input interlocks are satisfied.

DS Domain Description : The *DS* monitors many systems (e.g. early warning radar, GPS/INS positioning) to determine any threats to the aircraft. These systems are not pertinent to this problem so are abstracted through the statement:

> The *DS* is the defensive aids suite controller function which selects the flare type for release and if and when a flare is released in response to any threats it determines from its monitoring functions. The *DS* achieves this by formatting the flare selection and fire commands into its *cont* message which it transmits to the *DM*.

The *DM* knows the structure of the *cont* message it receives from the *DS*, so it can decode the *cont* message to determine the flare selection (*sel*) and fire (*fire*) components.

The justification for this problem progression is that:

$R4.1 \wedge Assumption(DS\_sel, cont) \Rightarrow R3.1.$

$R4.2 \wedge Assumption(DS\_fire, cont) \Rightarrow R3.2.$

$R4.3 \wedge Assumption(DS, DM) \Rightarrow R3.3.$

The $A_4 = Assumption(DS\_sel, DS\_fire, cont, DS, DM)$ is given by the *AS Domain Description.* Inspection of $R4.1$, $R4.2$ and $R4.3$ in conjunction with the *DS Domain Description* shows they imply $R3.1$, $R3.2$ and $R3.3$ as required. The constraints being relied upon is that *cont* contains the fire and select commands issued by the *DS*,and that the *DM* decodes them using *cont*.

This requirement transformation does not require additional information not already in the interface nor does it refer to future events. However it does rely on the useful behaviour from the domain being removed being "saved" as part of the assumption that justifies the transformation – it is an example of an Environment Constraint non-feasible requirement which is discussed in a POSE context in [83]. As noted above, this useful behaviour contribution is contained in the *DS Domain Description.*

The PPT application transforms the problem from P_{Red3} into P_{Red4}, and $R3$ is transformed into $R4$ (using the Requirements Interpretation part of the PPT) as follows:

R4.1 The *DM* shall decide which flare to select by observing a field in the *cont* message it receives.

R4.2 The *DM* shall decide to release a flare by observing a field in the *cont* message it receives.

R4.3 The selected flare shall only be released from the DU if the *DM* has received a flare release command when the input interlocks are satisfied.

R4.4 The three input interlocks are:

R4.4.1 *II* observes input st = air.

R4.4.2 *II* observes input pin = out.

R4.4.3 *II* observes pilot input of *allow*.

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

The domain *DS* is no longer referenced in the model and can be removed.

The software problem is transformed to P_{Red4}:

$$P_{Red4}: \begin{array}{l} DU(flare\_select, flare\_fire)_{fire,sel},\ II(allow, pin, st)^{int}, \\ DM(cont)^{sel,fire?},\ SC^{fire}_{int,fire?} \qquad \vdash\ R4^{flare\_select,flare\_fire}_{cont,st,pin,allow} \end{array}$$

An equivalent PF representation is shown in Figure H.7.

CLAIM: *The flare selection (DS_sel) and flare fire (DS_fire) commands are represented by the phenomena con.*

ARGUMENT & EVIDENCE: This is covered by the *DS Domain Description.*

PHENOMENA: None

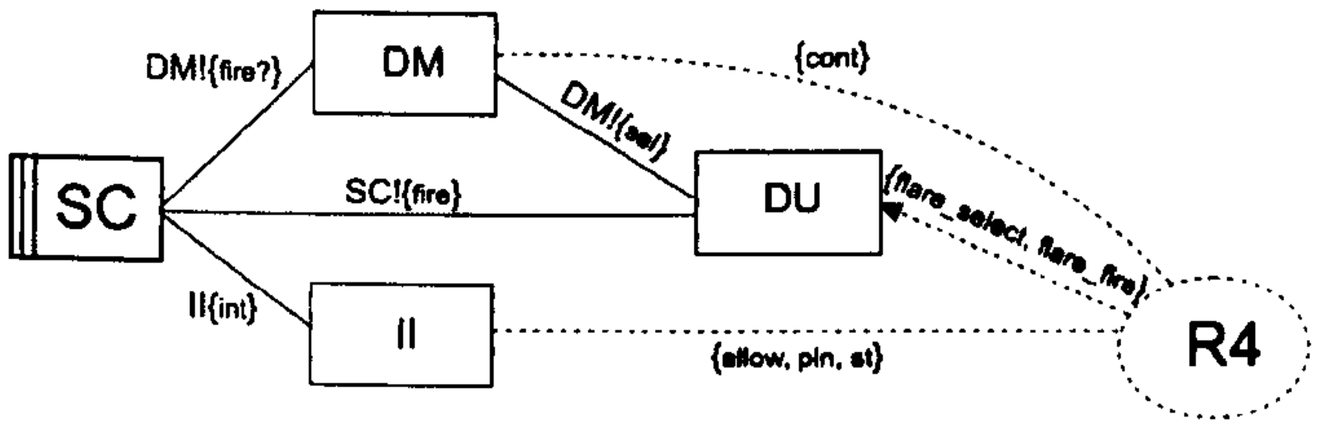

Figure H.7: Problem P_{Red4}
after *DS* Domain removal

H.3.5 PSA7 *DU* Domain Removal

The next domain to be removed is the Dispenser Unit domain *DU*, and the activities involved are captured in the PSA7 problem transformation step shown below.

PSA7: Application of the PPT for DU Domain Removal to problem P_{Red4} to form P_{Red}

INCLUDES: None

JUSTIFICATION *J*: The *DU* domain removal is based on the process given in section 5.1.2. The *DU* domain is adjacent to the requirement *R*4. The *DU* domain control phenomena are the *sel* flare selection input from the *DM* and the *fire* flare release command from the *SC*. These map directly to its output behaviour phenomena *flare_select* and *flare_fire* respectively. Therefore the effects *flare_select* and *flare_fire* will appear in the reference part of the requirement phenomena before the transformation, but will be replaced with their direct causes,*sel* and *fire*, after the transformation. The requirement *R*4 is as detailed in step PSA6 on page 303. Inspection of *R*4 indicates that only **R4.3** involves a reference to the *DU*.

Consider the transformation from **R4.3** into **R5.3** which involves the *DU* as noted above.

*R*4.3 : The selected flare shall only be released from the DU if the *DM* has received a flare release command and the input interlocks are satisfied.

*R*5.3 : The flare selected by the *DM sel* phenomena shall only be released by the *SC* issuing a *fire* command if the *DM* has received a flare release command and the input interlocks are satisfied.

DU Domain Description : The flare selected for release from the *DU* (*flare_sel*) is determined by the *sel* phenomena input from the *DM*. The *DU* will release the selected flare (*flare_fire*) when it receives a *fire* command input from the *SC*.

The justification for this problem progression is that:

$R5.3 \wedge Assumption(flare\_sel, flare\_fire, sel, fire) \Rightarrow R4.3.$

The $A_5 = Assumption(flare\_sel, flare\_fire, sel, fire)$ is given by the *DU Domain Description.* Inspection of $R5.3$ in conjunction with the *DU Domain Description* shows they imply $R4.3$ as required.

This requirement transformation does not require additional information not already in the interface nor does it refer to future events. However it does rely on the useful behaviour from the domain being removed being "saved" as part of the assumption that justifies the transformation – it is an example of an Environment Constraint non-feasible requirement which is discussed in a POSE context in [83]. As noted above, this useful behaviour contribution is contained in the *DU Domain Description.*

The PPT application transforms the problem from P_{Red4} into P_{Red}, and $R4$ is transformed into $R5$ (using the Requirements Interpretation part of the PPT) as follows:

R5.1 The *DM* shall decide which flare to select by observing a field in the *cont* message it receives.

R5.2 The *DM* shall decide to release a flare by observing a field in the *cont* message it receives.

R5.3 The flare selected by the *DM sel* phenomena shall only be released by the *SC* issuing a *fire* command if the *DM* has received a flare release command when the input interlocks are satisfied.

R5.4 The three input interlocks are:

R5.4.1 *II* observes input st = air.

R5.4.2 *II* observes input pin = out.

R5.4.3 *II* observes pilot input of *allow*.

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

The domain *DU* is no longer referenced in the model and can be removed.

The software problem is transformed to P_{Red}:

$$P_{Red}: \quad \begin{array}{l} II(allow, pin, st)^{int}, \\ DM(cont, sel)^{fire?},\ SC(fire)_{int,fire?} \end{array} \quad \vdash R5^{sel,fire}_{cont,st,pin,allow}$$

An equivalent PF representation is shown in Figure H.8.

PHENOMENA: None

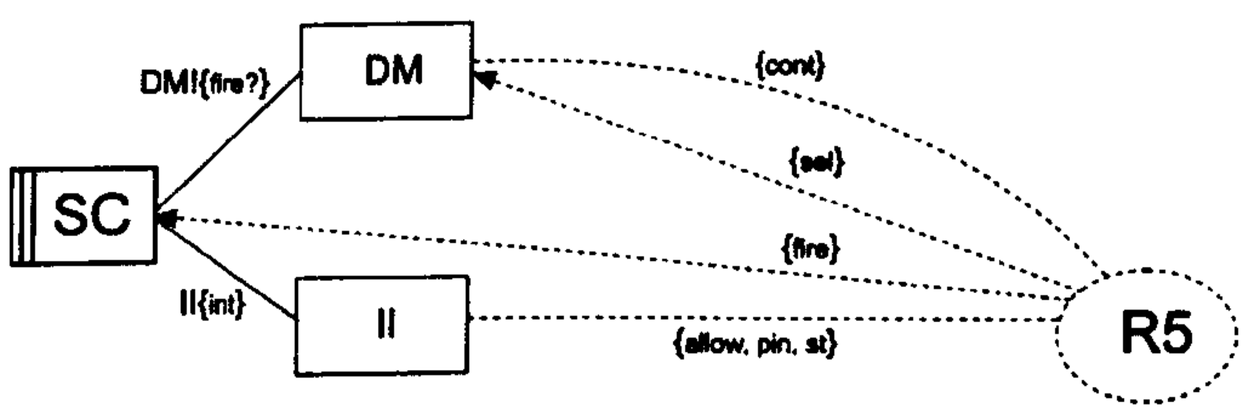

Figure H.8: Problem P_{Red}
after DU Domain removal

H.3.6 The PrePSA Step for the Revised DC Case Study

To derive an Alloy model from P_{Red} requires reconciliation of the requirements and domain phenomena, i.e., the relationship between them needs to be established. This could be achieved by applying the PPT to remove the DM and II domains to allow the specification to be derived, but as in the FA case study the preferred way forward (as the model is useful for the PSA) is to just apply the REQUIREMENTS INTERPRETATION transformation to extract the relationship between the phenomena. This can be captured as the PrePSA problem transformation step below, which forms the last part of the Pattern Problem Simplification task.

The PrePSA Step manages the transformation of P_{Red} into P_{Equiv}, which can be used to directly derive the specification of the SC and forms the basis of the PSA work.

PrePSA8: Application of REQUIREMENT INTERPRETATION *to problem* P_{Red} *to form* P_{Equiv}

CONCERNS: Reconciliation of the requirements and domain phenomena. The requirement $R5$ (associated with P_{Red}) refers to *cont*, *sel*, *fire*, *st*, *pin*, and *allow*, whilst the SC relates to *fire*?, *fire* and *int*. Therefore there is a need to apply REQUIREMENTS INTERPRETATION to identify the relationship between these phenomena, as in section 5.5.3.

JUSTIFICATION J: Inspection of the useful behaviour relation for the DM, II and SC domains indicate the logical equivalence of: *int* to *allow*, *pin* and *st*; *fire*? to *cont*; *fire* to *fire*. Further, the functionality of the SC does not involve flare selection, so the phenomena *sel* is not part of its specification.

This is justified by identifying the useful behaviour relations (domain contributions) of the DM, II and SC domains. The DM domain *effects* are concerned with decoding *cont* to extract the flare selection, *sel*, and whether a flare is to be released, *fire*?. The

DM sends the status of the fire request, *fire*?, to the *SC* which uses this information (in conjunction with *int*) to determine whether to output the flare release phenomena, *fire*. The *II* domain *effect* is to combine the input information it obtains from the phenomena *allow*, *pin* and *st*, and output this as the phenomena *int* sent to the *SC*.

The requirements *R*5 can be updated to *R*6 (the *SC* specification) as follows:

R6.1 Null

R6.2 The *SC* shall evaluate whether to release a flare by observing the *fire*? message it receives.

R6.3 A flare shall only be commanded to be released by the *SC* issuing a *fire* command if it has received a *fire*? message when the input interlocks are satisfied.

R6.4 The input interlocks are determined by *int*.

RS The *DC* shall mitigate **H1** & **H2** (Target: safety critical 10^{-7} fpfh).

These updates allow the problem to be transformed from P_{Red} into P_{Equiv} as follows.

$$P_{Equiv}: \quad II(int),\ DM(fire?),\ SC(fire)_{int,fire?} \qquad \vdash R6^{fire}_{int,fire?}$$

CLAIM: *The software problem P_{Equiv} allows the specification of SC to be derived.*

ARGUMENT & EVIDENCE: The requirement *R*6 uses the same phenomena as the domains, i.e., *R*6 is the specification.

CLAIM: *The requirements model is a valid representation of the system*

ARGUMENT & EVIDENCE: Informally, the *SC* receiving a valid *fire*? command when the *II* contribution, *int*, indicates the interlocks are satisfied combine to satisfy **R6.3**. Note that **R6.2** is part of **R6.3**. The *II* contribution covers **R6.4**. The PSA considers **RS**. Therefore, (informally) the POSE requirements model (P_{Equiv}) satisfies the requirement (specification) *R*6. This will be formally checked as part of the Alloy simulation and proof work.

H.4 POSE Safety Pattern Activity 4

The fourth activity in the POSE safety pattern is to perform the safety analysis tasks required by the PSA.

The validation and safety analysis (PSA) of the case study makes use of formal modelling using Alloy and is based on the four part process summarised in section 6.4:

1. Derive the Alloy model directly from the POSE software problem description – using a template.

2. Use the following four-part process for applying the formal modelling:

   (a) use simulation to develop the model and gain confidence it has the required behaviour;

   (b) perform formal proof to validate the model;

   (c) use the model to prove safety properties (if necessary)

   (d) investigate any specific problem areas using simulation. (This process is efficient because the more expensive proof work is only undertaken once simulation has provided confidence in the efficacy of the model.)

3. Use the FFA/FTA and modified HAZOPS form for domains not formally modelled.

The formal analysis was conducted using a variety of Alloy files to cover simulation, proof of requirements and investigation of the model's properties. The simulation and proof results were used to develop an in-depth understanding of the model and guided its development as follows.

H.4.1 *DC* Model Simulation

The initial model of the *DC* system used a simple version of the *SC* which is shown below

```
### Define Behaviour
pred SCBehave [t,t':Time]{ //R6.3
  (II.int1.t = val && DM.fireq.t = rel) => SC.fire.t' = rel
  else SC.fire.t' = noac }
```

When doing the simulation with the above simple model for the *SC*, the results showed that *SC.fire* was set to *rel* in the next time instant as long as the interlocks were valid ($II.int1 = val$) and *DM.fireq* was *rel* in the current time instant. This allowed *DM.fireq* to be *rel* at the same time instant as $II.int1 = val$. However, the intention was to improve integrity by requiring that *DM.fireq* should be the initiating event for flare release – that is, the interlocks should become (and remain) valid prior to *DM.fireq* being set to *rel*, other combinations should inhibit flare release. The specification *R*6 and the requirement *R* do not specify this explicitly and a derived requirement *RD* was introduced to cover this desired behaviour:

RD:

> The *SC* shall only send the *fire* message if it has received an active *fireq* with the interlocks being valid. The interlocks must be valid prior to the arrival of *fireq* and continue to be valid through the flare release process.

The specification was modified to become *R*6+ ($R6+ = R6 + RD$).

H.4.2 *DC* Model Formal Proof

The formal proof of requirements to validate the model used the Alloy model shown below. The assert statements R63full, R51, R52, R53full and R54 validate that the specification R6.3, and the requirements R5.1, R5.2, R5.3 and R5.4 respectively, are satisfied by the model.

```
//### Problem::=  Decoy Controller: DC
// Model based on R5/PRed/Figure 2.7 -- Reduced model including sel behaviour.

open util/ordering[Time] as Ti
sig Time{}

//### Define Phenomena
sig FIR {}
one sig noac, rel extends FIR {}        // Fire command has two states; no action or fire.

sig INL {}
one sig val, nov extends INL {}         // Status of II.int.t is either valid or not valid.

sig CONT {}
one sig c0, c1, c2, c3 extends CONT {}  // c0 is default and selects s0; c1 and c3 select s1 and s3;
                                        // c2 is a fire command, leaves flare selection unchanged.
sig SEL {}
one sig  s0, s1, s3 extends SEL {}      // Flare selection: s0 is default, s1 and s3 are other flare types

sig AIR {}
one sig  air, gnd extends AIR {}        // Aircraft status, either in the air or on the ground.

sig PIN {}
one sig  out, inn extends PIN {}        // Safety pin is either in or out.

sig OK {}                                       // Used for PILOT Allow choice and others
one sig  yes, noo extends OK {}


//### Define Domains
one sig SC { fire : FIR -> Time} {}
one sig DM { sel : SEL->Time, fireq: FIR -> Time} {}
one sig II { int1 : INL -> Time} {}

// EXT represents the external interface of the model
one sig EXT { cont : CONT -> Time,  allow : OK->Time, pin : PIN->Time, st : AIR->Time} {}


pred SCBehave [t,t':Time]{ let tp = prev[t] |       //R5.3 + RD
   (II.int1.tp = val && II.int1.t = val && DM.fireq.t = rel) => SC.fire.t' = rel
   else SC.fire.t' = noac }
```

```
pred DMBehave [t,t':Time]{
//R5.1
( EXT.cont.t = c0 => DM.sel.t' = s0
 else  EXT.cont.t = c1 => DM.sel.t' = s1
          else  EXT.cont.t = c2 => DM.sel.t' = DM.sel.t
                   else EXT.cont.t = c3 => DM.sel.t' = s3)
&&
//R5.2
( EXT.cont.t = c0 => DM.fireq.t' = noac
 else  EXT.cont.t = c1 => DM.fireq.t' = noac
          else  EXT.cont.t = c2 => DM.fireq.t' = rel
                   else EXT.cont.t = c3 => DM.fireq.t' = noac)}

pred IIBehave [t,t':Time]{ //R5.4
 ((EXT.allow.t = yes && EXT.pin.t = out && EXT.st.t = air) => II.int1.t' = val
    else II.int1.t' = nov )}

// EXT version set to support proof work
pred EXTBehave [t,t':Time]{
(EXT.cont.t' = c0 or EXT.cont.t' = c1 or  EXT.cont.t' = c2 or  EXT.cont.t' = c3 ) &&
(EXT.allow.t' = noo or EXT.allow.t' = yes) &&
( EXT.pin.t' = out or EXT.pin.t' = inn)  &&
(EXT.st.t' = gnd or EXT.st.t' = air)}


// Predicate trange returns true if t is in the range from ts to tf.
pred trange [t : Time, ts, tf : Int] { #prevs[t] >= ts && #prevs[t] =< tf}

// Predicate change returns true if the phenomena a has changed state.
fun change [a : FIR -> Time, t1 : Time] : OK {a.t1 != a.(prev[t1]) => yes else noo}

// Predicate timeorder returns true if the times t1,t2, and t3 are in an ascending sequence.
fun timeorder [t1,t2,t3 : Time] : OK {
(t1 = prev[t2] && t2 = prev[t3] && t1 = prev[prev[t3]])  => yes else noo}


//### Define the Trace Model

pred init [t : Time] {SC.fire.t = noac &&  DM.sel.t = s0 && DM.fireq.t = noac && II.int1.t=nov &&
                                EXT.cont.t = c0 && EXT.allow.t = noo && EXT.pin.t = out && EXT.st.t=gnd}
```

```
fact traces { init[Ti/first[]]
   all t : Time - Ti/last[] | let t' = Ti/next[t] |  SCBehave[t,t'] && DMBehave[t,t']  &&
                                                     IIBehave[t,t'] && EXTBehave[t,t'] }
```

```
//## Investigate Properties

assert R63full {all t1,t2,t3 : Time |
(timeorder[t1,t2,t3]=yes && II.int1.t1=val && II.int1.t2=val && DM.fireq.t2=rel => SC.fire.t3=rel)
&&
(timeorder[t1,t2,t3]=yes && not(II.int1.t1=val && II.int1.t2=val && DM.fireq.t2=rel) => SC.fire.t3=noac)}
```

```
assert R51 {all t1,t2,t3 : Time |
( timeorder[t1,t2,t3] = yes && EXT.cont.t1 = c0 => DM.sel.t2 = s0) &&
( timeorder[t1,t2,t3] = yes && EXT.cont.t1 = c1 => DM.sel.t2 = s1) &&
( timeorder[t1,t2,t3] = yes && EXT.cont.t1 = c3 => DM.sel.t2 = s3) }
```

```
assert R52 {all t1,t2,t3 : Time |
( timeorder[t1,t2,t3] = yes && EXT.cont.t1 = c0 => DM.fireq.t2 = noac) &&
( timeorder[t1,t2,t3] = yes && EXT.cont.t1 = c1 => DM.fireq.t2 = noac) &&
( timeorder[t1,t2,t3] = yes && EXT.cont.t1 = c2 => DM.fireq.t2 = rel) &&
( timeorder[t1,t2,t3] = yes && EXT.cont.t1 = c3 => DM.fireq.t2 = noac) }
```

```
assert R53full {all t1,t2,t3 : Time |
(timeorder[t1,t2,t3]=yes && DM.sel.t2=s0 && II.int1.t1=val && II.int1.t2=val && DM.fireq.t2=rel &&
    EXT.cont.t1 = c2 => (SC.fire.t3 = rel ) ) &&

(timeorder[t1,t2,t3]=yes && DM.sel.t2=s1 && II.int1.t1=val && II.int1.t2=val && DM.fireq.t2=rel &&
    EXT.cont.t1 = c2 => (SC.fire.t3 = rel ) ) &&
(timeorder[t1,t2,t3]=yes && DM.sel.t2=s3 && II.int1.t1=val && II.int1.t2=val && DM.fireq.t2=rel &&
    EXT.cont.t1 = c2 => (SC.fire.t3 = rel ) )
&&
(timeorder[t1,t2,t3]=yes && not(II.int1.t1=val && II.int1.t2=val && DM.fireq.t2=rel) => SC.fire.t3=noac )}
```

```
assert R54 {all t1,t2,t3 : Time |
(timeorder[t1,t2,t3]=yes && EXT.allow.t1=yes && EXT.pin.t1=out && EXT.st.t1=air => II.int1.t2=val) &&
(timeorder[t1,t2,t3]=yes && not(EXT.allow.t1=yes && EXT.pin.t1=out && EXT.st.t1=air) => II.int1.t2=nov)}
```

```
pred show1[] {}

run show1 for 24 but 6 int

check R63full for 24 but 6 int

check R51 for 24 but 6 int
check R52 for 24 but 6 int
check R53full for 24 but 6 int
check R54 for 24 but 6 int
```

H.4.3 *DC* Model Safety Analysis

The validation proof for the Alloy model assert R53full (refer to the Alloy model above) demonstrates that the system represented by the specification *R*6+ (including derived requirement *RD*) provides mitigation for **H1** & **H2**.

The safety analysis was based on the FFA/HAZOPS/FTA approach and followed the work presented in section 5.2.2. Table 5.3 contains the summary for the HAZOPS and the entries in this table were investigated further using the simulation model, the results of this work are presented in the next section.

H.4.4 *DC* Model Investigate Problem Areas

The simulation of the entries in Table 5.3 was performed using the timing model introduced in section 5.5.4. The external inputs for the safety model (based on *R*5) were modelled using the domain *EXT* and the input timing was varied using the structure shown below.

```
// EXT version set to support simulation work
pred EXTBehave [t,t':Time]{
  (trange[t,0,2] => EXT.cont.t' = c0
    else trange[t,3,5] =>  EXT.cont.t' =  c2
        else trange[t,6,8] =>  EXT.cont.t' = c3
            else trange[t,9,10] =>  EXT.cont.t' =  c1
```

```
                    else trange[t,11,21] =>  EXT.cont.t' = c2
                         else trange[t,22,24] =>  EXT.cont.t' = c0)
 &&
(trange[t,0,1] => EXT.allow.t' = noo
  else trange[t,2,5] =>  EXT.allow.t' =  yes
       else trange[t,6,8] =>  EXT.allow.t' = noo
            else trange[t,9,11] =>  EXT.allow.t' =  noo
                 else trange[t,12,18] =>  EXT.allow.t' = yes
                      else trange[t,19,24] =>  EXT.allow.t' = noo)
 &&
(trange[t,0,1] => EXT.pin.t' = inn
  else trange[t,2,5] =>  EXT.pin.t' =  out
       else trange[t,6,8] =>  EXT.pin.t' = out
            else trange[t,9,11] =>  EXT.pin.t' =  out
                 else trange[t,12,19] =>  EXT.pin.t' = out
                      else trange[t,20,24] =>  EXT.pin.t' = inn)
 &&
(trange[t,0,1] => EXT.st.t' = gnd
  else trange[t,2,5] =>  EXT.st.t' =  air
       else trange[t,6,8] =>  EXT.st.t' = air
            else trange[t,9,11] =>  EXT.st.t' =  air
                 else trange[t,12,20] =>  EXT.st.t' = air
                      else trange[t,21,24] =>  EXT.st.t' = gnd) }
```

The simulation work validated the outcomes defined in Table 5.3, but also found some problematic behaviour. The simulation for the cases where continuous *fireq* (*fire*? in Table 5.3) was received, H4 and H5, indicated that a flare release could occur whenever consecutive periods of $II.int1 = val$ were encountered – which is the outcome outlined in section 4.4. The current specification ($R6+$) assumes that *fireq* is a single initiating entity of relatively short duration, but this is not the case when a failure such as H4 occurs. The integrity of the model needs to be further improved to mitigate the case where *fireq* has failed active by adding an additional derived requirement as follows:

RDD:

> The *SC* shall only send the *fire* message if it detects a transition of *fireq* going from inactive (*noac*) to active (*rel*) with the interlocks being

valid prior to the arrival of *fireq* going active and continuing to be valid through the flare release process.

The behaviour of *SC* in the model (SCBehave) was modified as follows to incorporate the derived requirement *RDD*:

```
pred SCBehave [t,t':Time]{ let tp = prev[t] |      //R5.3 + RDD
   (II.int1.tp = val && II.int1.t = val && DM.fireq.tp = noac && DM.fireq.t = rel)
        => SC.fire.t' = rel   else SC.fire.t' = noac }
```

The simulation was repeated and the model behaved correctly with respect to dealing with *fireq* failing active. Therefore the derived requirement *RDD* was added to replace *RD*, and this formed the specification $R6{+}{+} = R6{+}RDD$. The validation proofs of R5.1, R5.2 and R5.4 were repeated successfully on the revised model. The assert statements for R6.3 and R5.3 were amended to reflect *RDD* and successfully proved for the model – the revised assert for *R*6.3 is shown below.

```
assert R63full {all t1,t2,t3 : Time |
(timeorder[t1,t2,t3]=yes && II.int1.t1=val && II.int1.t2=val && DM.fireq.t1=noac  && DM.fireq.t2=rel
   => SC.fire.t3 = rel )
&&
(timeorder[t1,t2,t3]=yes && not(II.int1.t1=val && II.int1.t2=val && DM.fireq.t1=noac && DM.fireq.t2=rel)
   => SC.fire.t3 = noac ) }
```

H.5 Revised *DC* Case Study Discussion

The revised *DC* case study has demonstrated another successful application of the POSE safety pattern using the "good" practice requirements form based on [118] introduced in section 4.1.1).

The case study provided further support for the properties **Simplification**, **[NecessaryP]** and mitigation of the issue **[LateI3]**. Support for the property **Simplification** was demonstrated through the problem progression steps in section H.3.1 through H.3.5, whilst support for **[NecessaryP]** was demonstrated through

the derivation of the specification in section H.3.6. Mitigation of the issue **[LateI3]** was provided by the safety analysis work covered in section H.4.

The case study again demonstrated the successful use of the four part formal POSE/Alloy modelling process summarised in section 6.4. The modelling identified a shortfall in the safety integrity which was rectified by augmenting the requirements with a derived requirement and then validated using the process. The problem was that the requirement $R6.3$ was not specific enough to exclude unwanted behaviours and it allowed a flare release even when the interlock became valid at the same time as the fire command had been received – this was corrected by defining the derived requirement RD. The approach also identified the problem with *fireq* failing active and this was resolved by adding the derived requirement RDD to replace RD to form the specification $R6++$.

The modelling process allows the behaviour of the proposed development to be investigated in detail, and any shortcomings highlighted and resolved. In this case study the requirement R (see section H.1) and specification $R6$ were improved by the definition of the derived requirement RDD to clarify the desired behaviour.

Appendix I

Analysis of Research Contribution

The goal of this appendix is to present a more detailed analysis of my research contribution on using POSE in the early stages of a safety critical system development, as an extension of the work presented in section 7.5. As noted in noted in section 7.5 my contribution has three main parts:

- developing and extending the theory of Problem Orientation;
- applying the extended theory in a safety critical engineering setting;
- providing partial validation for the extended theory within that setting.

These three elements of my contribution are depicted in Figure I.1 and each is expanded further in the sections that follow.

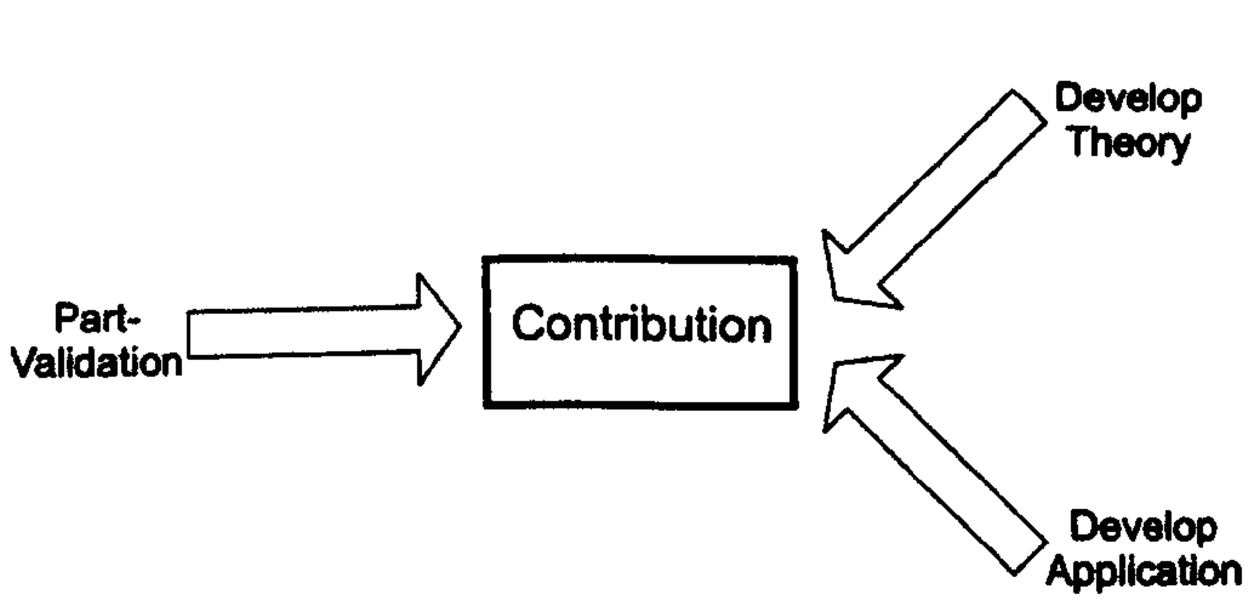

Figure I.1: Contribution Summary

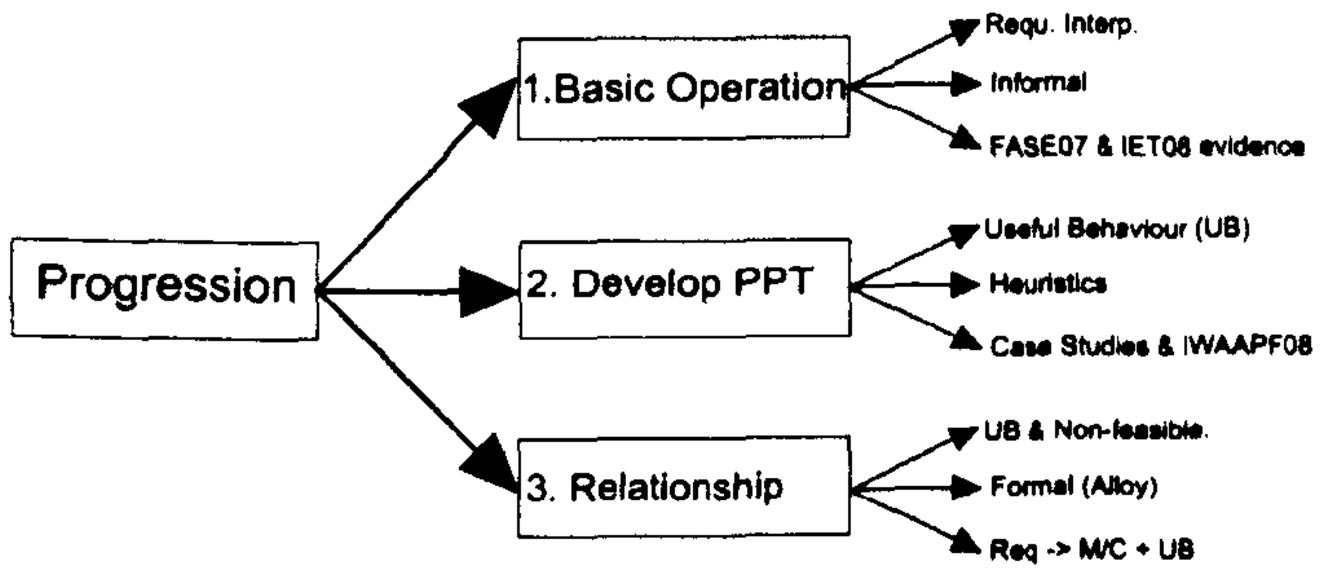

Figure I.2: PPT for Safe Systems Development

I.1 Detailed Discussion of Develop the Theory

I.1.1 Develop the Theory Detailed 1 - Problem Progression

The problem progression transformations were raw and relatively undeveloped, and I developed – in parallel with other work in this by Jon Hall, Lucia Rapanotti and Zhi Li – a process for applying them (as published in the FASE07 [87] paper). This culminated in the POSE problem progression for safety systems discussed in section 5.1.2 – the basic elements of this development are depicted in Figure I.2.

A contribution to knowledge is the resulting engineering form of the PPT and a method to support its application (IWAAPF08 [83] paper , and section 5.1.2). Also, this research allowed me to comment further on the relationship between the notion of a domain's useful behaviour and the work on non-implementable requirements from Zave and Jackson's Four Dark Corners paper [133] - as discussed in the IWAAPF08 [83] paper, and presented in section 5.1.1.1 and section 5.1.1.2.

As noted, another contribution is a collection of useful heuristics for engineers for its application (IWAAPF08 [83] paper, and section 5.1.2) that substantiate the practical nature of Jackson's original idea sketch. These were used extensively in the case studies that followed section 5.1.2.

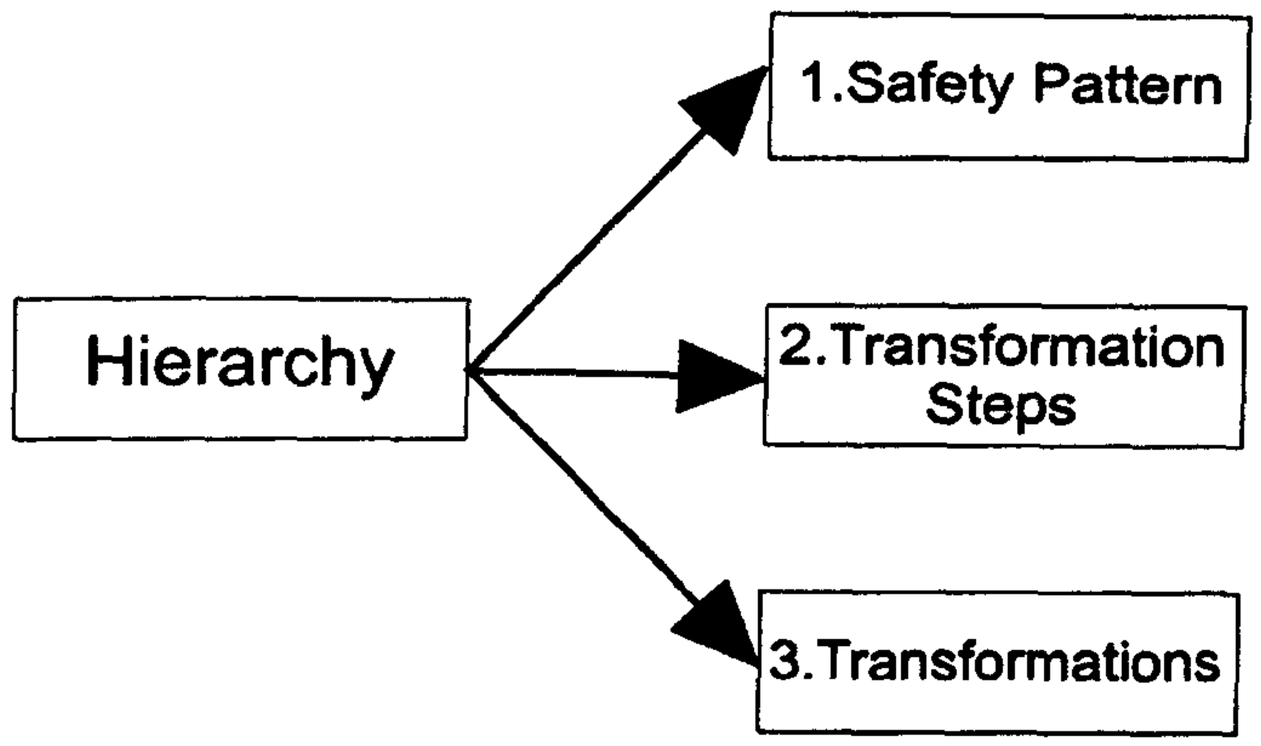

Figure I.3: POSE Abstraction Hierarchy

I.1.2 Develop the Theory Detailed 2 – How to Sequence

Another contribution to knowledge is the ability to sequence the transformations through the abstraction provided by the POSE safety pattern (FASE07 [87], SSS08 [88], and in section 6.1) and the use of problem transformation steps (refer to section 4.1.4.2) which collect basic POSE transformations into meaningful engineering steps in the development process - and allows appropriate validation at that level. The resulting hierarchy is shown in Figure I.3.

This abstraction hierarchy is useful for managing development tasks. Further, this work was extended by my supervisors to be the POE Process Pattern (Assurance-Driven Development in POE [37]).

I.2 Detailed Discussion of Application

I.2.1 Application 1

Develop the applicability aspects of POSE using the case studies as exemplars. The case studies contribute to knowledge in a number of ways, e.g. they provided evidence to demonstrate that an extended POSE is a capable SSSD (Safe Software System Development) approach. This contribution is phrased in terms of 10 identified properties (Table 2.3), which showed that, from a practical engineering perspective,

extended POSE was a suitable SSSD approach.

The applicability aspect was also explored with respect to the Research Objectives (refer to section 1.2), the results being that:

1. a suitable specification can be derived using the PPT (FASE07, IWAAPF08, and sections 5.1.3, 5.5.3 and 6.3.3 etc.);

2. the PrePSA step (e.g. page 168 and page 180) was developed to support deriving the specification;

3. the safety analysis involving HAZOPS could be enhanced using the structure and information provided by POSE (refer to section 5.2);

4. the traceability and iteration capabilities under POSE assist backtracking (extension of HASE07 first presented in IET07, and developed in section 5.3);

5. analysis under POSE is efficient because the model used for the safety analysis was the same as that used in the development (e.g. SSS08 and section 7.2.8).

Further, the POSE safety pattern and the satisfaction of the property **Context** satisfies the research objective of capturing and separating contextual knowledge from the design of the system and its desirable properties.

I.2.2 Application 2

The assessment was conducted against six issues and four properties, which were identified by analysis and experience of using IPDP (refer to section 3.4.5) and evidenced from the (retrospective) case studies – *DC* (introduced in section 4.1.1), *SMS* (introduced in section 5.5.1) and *FAS* (introduced in section 6.3). As part of this contribution, we have argued that, as validation and verification (V&V) are an important aspect of the safety standards, the extended POSE caters for them at a number of levels: all transformations are justified, the problem transformation steps

include obligations that have to be discharged and the POSE safety pattern includes the PSA step (FASE07, SAFECOMP08 and section 6.1). The POSE safety pattern and the problem transformation steps (e.g. PS1 in section 4.2) provide structuring that supports the objective of keeping track of safety artefacts for validation and certification purposes.

As a final contribution in this section, this research provides evidence that POSE can be used with formal notations to provide formal proof and simulation capabilities (e.g. SSS08, section 6.3.4, section 6.3.6 and section 6.3.7) including the following developments:

1. Techniques for forming Alloy model directly from POSE model (e.g. section 5.5.4 and section 6.3.4.1)

2. Development of the Timing model (e.g. section 5.5.4)

3. Proposed means of performing formal analysis based on 4-step process (refer to section 6.4).

The work in section 5.2 and the subsequent case study results demonstrated that the extended POSE satisfies the research objective concerning the allowing for both formal and informal safety analysis techniques to take place.

I.3 Partial Validation

The three, varied case studies provide some evidence to support and validate the engineering application, as does the industrial use reported in the thesis (refer to section 7.3). They also show that extended POSE integrates well with the IPDP and satisfies the provisions of a number of important safety standards and guidelines (e.g. DS 00-56 [125], IEC61508 [51] and DO-178B [108]) providing support for the integrating with current safety engineering practice research objective.

The case studies were retrospective so it is always difficult to counter the argument that hindsight rather than the capabilities of the approach were responsible for finding the issues. To counter this hindsight issue, the case studies were organised to integrate with the IPDP and follow its standard course – care was taken to avoid performing analyses which were known to be capable of detecting the issues if they were not part of the standard approach or were not part of the extensions provided by using POSE. For example, the requirements were based closely on those used with the original development and were not (apart from the work reported in Appendix H on the revised *DC* case study) "improved" to a more idealised form. The ongoing industrial work is not retrospective and is proceeding with some success – providing good validation of the approach and demonstrating that extended POSE does improve the capabilities of the early phases of IPDP.

I.4 Thesis Contribution Summary

The POSE safety pattern and its associated case study work demonstrate that extended POSE does support the integration of safety analysis into the early phases of a system development, which satisfies the first research objective listed in section 1.2.

In summary POSE, as extended by the research work reported in this thesis, does satisfy the research objectives listed in section 1.2 of:

1. integrating safety analysis within the early phases of development, particularly requirement analysis and high level architectural design (this section)

2. allowing for both formal and informal safety analysis techniques to take place (section I.2.2)

3. capturing and separating contextual knowledge from the design of the system and its desirable properties (section I.2.1)

4. keeping track of safety artefacts for validation and certification purposes (section I.2.2)

5. integrating with current safety engineering practice (section I.3).

The work in this thesis has also demonstrated that extended POSE can be used within IPDP to improve the capabilities of IPDP – thus satisfying the second objective from section 1.2 of improving the early phases of an existing safety development process (section I.3).

Bibliography

[1] M. A. D. Abdala, C. Lahoz, and R. de Lemos. Diversity of Safety Arguments in the Validation of a Sounding Rocket Destruction System. In *19th International System Safety Conference*, Huntsville, Alabama, USA, 2001. System Safety Society.

[2] P. A. Abdulla, J. Deneux, G. Stålmarck, H. Ågren, and O. Åkerlund. Designing Safe, Reliable Systems Using Scade. In *ISoLA 04, LNCS 4313*, pages 115–129, 2006.

[3] Civil Aviation Authority Air Traffic Services. CAP 670 ATS Safety Requirements. Technical report, 1998.

[4] O. Åkerlund, S. Nadjm-Tehrani, and G. Stalmarck. Integration of Formal Methods into System Safety and Reliability Analysis. In *17th International System Safety Conference*, Orlando, Florida, USA, 1999. System Safety Society.

[5] ASSERT. Project Website: http://www.assert-project.net/. Technical report, 2008.

[6] Chemical Industries Association. Guide to Hazard and Operability Studies. Report CISHEC/8906/1000, CIA, 1977.

[7] J. Barnes. *High Integrity Ada: The SPARK Approach.* Addison-Wesley, 1997.

[8] I. Bate and P. Conmy. Safe Composition of Real Time Software. In *Proceedings of the Ninth International Symposium on High-Assurance Systems Engineering HASE'05*, Heidelberg, Germany, October 2005.

[9] R. Bharadwaj and C. Heitmeyer. Developing high assurance avionics systems with the SCR requirements method. In *Proceedings the 19th DASC*, volume 1, 2000.

[10] R. Bharadwaj and S. Sims. Combining constraint solvers with BDDs for automatic invariant checking. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2000*, Berlin, Germany, March 2000.

[11] P. Bieber, JP. Blanquart, G. Durrieu, J. Lucotte, F. Tardy, M. Turin, C. Seguin, and E. Conquet. Integration of formal fault analysis in ASSERT: Case studies and lessons learned. In *European Congress on Embedded Real-Time Software ERTS 2008, SIA, AAAF, SEE*, Toulouse, France, 2008.

[12] R Bloomfield, P. Bishop, C. Jones, and P. Froome. ASCAD - Adelard Safety Case Development Manual. Technical report, 1998.

[13] G. Booch. *Object-Oriented Analysis and Design with Applications.* Benjamin/Cummings, Redwood City, CA, 2nd edition, 1994.

[14] P. Caseley, N. Tudor, and C. OíHalloran. The case for an evidence based approach to software certification. Technical report, Safety standards review committee, UK Ministry of Defence, 2003.

[15] CCMB-2006-09-001. Common Criteria for Information Technology Security Evaluation Part 1: Introduction and General Model. Ver. 3.1 Rev.1, September 2006.

[16] P. Checkland. *Systems Thinking, Systems Practice.* John Wiley & Sons, 1981.

[17] C. Choppy and M. Heisel. Use of Patterns in Formal Development: Systematic Transition from Problems to Architectural Designs. In *16th International Workshop on Algebraic Development Techniques, WADT 2002, LNCS 2755*, pages 201–215, Berlin, September 2002. Springer-Verlag.

[18] Richard Colgren. *Basic MATLAB(R), Simulink(R), and Stateflow(R) (AIAA Education Series)*. AIAA, 1st edition, 2006.

[19] P.-J. Courtois and D. L. Parnas. Documentation for Safety Critical Software. In *15th International Conference on Software Engineering*, pages 315–323, Baltimore, USA, 1997.

[20] Karl Cox, Jon G. Hall, and Lucia Rapanotti, editors. *Proceedings of ICSE 1st International Workshop on Applications and Advances of Problem Frames*. IEEE CS Press, 2004.

[21] Karl Cox, Jon G. Hall, and Lucia Rapanotti, editors. *Journal of Information and Software Technology: Special issue on Problem Frames*, volume 47. Elsevier, November 2005.

[22] Lord Cullen. The public inquiry into the Piper Alpha disaster. Technical Report Cm 1310, HMSO, 1990.

[23] M.F. da Cruz and P. Raistrick. AMBERS: Improving Requirements Specification Through Assertive Models and SCADE/DOORS Integration. In *Proceedings of the Safety-critical Systems Symposium*, Bristol, UK, 2007. Springer-Verlag.

[24] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993.

[25] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. GRAIL/KAOS: an environment for goal-driven requirements engineering. In *ICSE '97: Pro-*

ceedings of the 19th international conference on Software engineering, pages 612–613, New York, NY, USA, 1997. ACM.

[26] R. de Lemos, A. Saeed, and T. Anderson. On the Integration of Requirements Analysis and Safety Analysis for Safety-Critical Systems. Technical Report CS-TR:629, http://citeseer.ist.psu.edu/536230.html, University of Newcastle upon Tyne, 1998.

[27] DoD. MIL-STD-1629A, Procedures for Performing a Failure Mode, Effects and Criticality Analysis. Technical report, 1984.

[28] DoD. *MIL-STD-882D Standard Practice for System Safety.* 10 February 2000.

[29] A. F. Ellis. Achieving Safety in Complex Control Systems. In Tom Anderson and Felix Redmill, editors, *Achievement and Assurance of Safety, Safety-critical Systems Symposium*, Brighton, UK, 1995. Springer-Verlag.

[30] C. A. Ericson II and S. S. Ting. Design for System Safe Software. In *20th International System Safety Conference*, 2002.

[31] Esterel. Getting Started with SCADE. In Tom Anderson and Felix Redmill, editors, *Proceedings of the Safety-critical Systems Symposium*, Bristol, UK, 2007. Springer-Verlag. http://www.estereltechnologies.com/technology/getting-started/scade.

[32] Andy Galloway, Frantz Iwu, John McDermid, and Ian Toyn. On the Formal Development of Safety Critical Software. In *Verified Software, LNCS 4171*, pages 362–373, Berlin, 2008. Springer-Verlag.

[33] A. Gerstinger, G. Schedl, and W. Winkelbauer. Safety versus Reliability: Different or Equal. In *20th International System Safety Conference*, pages 393–400, Denver, Colorado, USA, 2002. System Safety Society.

[34] P. J. Graydon, E. A. Strunk, and J. C. Knight. Assurance Based Development of Critical Systems. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 347–357, June 2007.

[35] J. G. Hall, D. Mannering, and L. Rapanotti. Arguing safety with Problem Oriented Software Engineering. In *HASE'07*, pages 23–32. IEEE, 2007.

[36] J. G. Hall and L. Rapanotti. A framework for software problem analysis. Technical report 2005/05, Open University, 2005.

[37] J. G. Hall and L. Rapanotti. Assurance-driven development in problem oriented engineering. Technical Report 2007/15, Computing Dept., The Open University, 2007.

[38] J. G. Hall and L. Rapanotti. POELog: a Prolog-based engine for Problem Oriented Engineering. Technical Report 2008/07, Open University, Dept. of Computing, 2008.

[39] J. G. Hall, L. Rapanotti, and M. A. Jackson. Problem Oriented Software Engineering. Technical Report 2006/10, Open University, Dept. of Computing, 2006.

[40] J. G. Hall, L. Rapanotti, and M. A. Jackson. Problem Oriented Software Engineering: A design-theoretic framework for software engineering. In *5th IEEE Int. Conference on Software Engineering and Formal Methods (SEFM 2007)*. IEEE, 2007.

[41] J. G. Hall, L. Rapanotti, and M. A. Jackson. Problem Oriented Software Engineering: Solving the Package Router Control Problem. *Software Engineering, IEEE Transactions on*, 34(2):226–241, March-April 2008.

[42] J. Hammond, R. Rawlings, and A. Hall. Will it Work? In *5th IEEE International Symposium on Requirements Engineering (RE'01)*, pages 102–109, August 2001.

[43] I. Hayes, M. A. Jackson, and C. Jones. Determining the Specification of a Control System from That of Its Environment. In *FME '2003', LNCS 2805*, pages 154–169, Berlin, 2003. Springer-Verlag.

[44] M. Heisel and J. Souquieres. A Method for Requirements Elicitation and Formal Specification. In *Conceptual Modeling – ER'99, LNCS 1728*, pages 309–324, Berlin, 1999. Springer-Verlag.

[45] C. Heitmeyer, M. Archer, R. Bharadwaj, and R. Jeffords. Tools for constructing requirements specifications: The SCR toolset at the age of ten. *International Journal of Computer Systems Science & Engineering*, vol. 1:19–35, 2005.

[46] C. Heitmeyer and R. Jeffords. Applying a formal requirements method to three NASA systems: Lessons learned. In *IEEE Aerospace Conference*, Big Sky, MT, 2007.

[47] C. Heitmeyer, J. Kirby Jr., B. Labaw, M. Archer, and R. Bharadwaj. Using abstraction and model checking to detect safety violations in requirements specifications. *IEEE Transactions on Software Engineering*, vol. 24:927–948, 1998.

[48] C. Heitmeyer, D. Kiskis, and B. Labaw. Consistency checking of SCR-style requirements specifications. In *Second IEEE International Symposium on Requirements Engineering*, 1995.

[49] A. J. Hilton and J. G. Hall. Developing critical systems with PLD components. In Tiziana Margaria and Mieke Massink, editors, *FMICS '05: Proceedings*

of the 10th international workshop on Formal methods for industrial critical systems, pages 72–79, Lisbon, Portugal, 2005. ACM Press.

[50] Sophie Humbert, Christel Seguin, Charles Castel, and Jean-Marc Bosc. Deriving Safety Software Requirements from an AltaRica System Model. In *27th International Conference on Computer Safety, Reliability and Security, LNCS 5219*, pages 320–331. Springer-Verlag, 2008.

[51] International Electrotechnical Commission. Functional safety of electrical/electronic/programmable electronic safety-related systems. Technical Report IEC 61508.

[52] D. Jackson. *Software Abstractions Logic, Language & Analysis.* MIT Press, 2006.

[53] M. A. Jackson. *Problem frames: analysing and structuring software development problems.* Addison-Wesley, Harlow, 2001.

[54] M. A. Jackson. Why software writing is hard and will remain so. *Inf. Process. Lett.*, 88(1-2):13–25, 2003.

[55] M. A. Jackson. Problem Frames and Software Engineering. *Information and Software Technology*, 47(14):903–912, 2005.

[56] I. Jacobson. *Object Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, Wokingham, UK, 1992.

[57] Cliff B. Jones. Specification Before Satisfaction: The Case for Research into Obtaining the Right Specification. In *4th International Conference of Z and B users, ZB 2005, LNCS 3455*, pages 1–5, Berlin, April 2005. Springer-Verlag.

[58] A. Joshi and P. E. Heimdahl. Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier. In *SAFECOMP 2005, LNCS 3688*, pages 122–135, Berlin, 2005. Springer-Verlag.

[59] T. P. Kelly. *Arguing safety - A systematic approach.* PhD thesis, Department of Computer Science,University of York, 1998.

[60] T. P. Kelly. A systematic approach to safety case management. In *SAE 2004 World Congress*, Detroit, USA, 2004.

[61] T. P. Kelly, J. A. McDermid, and R. Weaver. Goal-based Safety Standards: Opportunities and Challenges. In *23rd International System Safety Conference*, pages 393–400, San Diego, USA, 2005. System Safety Society.

[62] T. P. Kelly and R. Weaver. The goal structuring notation - a safety argument notation. In *AssWS. Workshop on assurance cases: Best practices, possible obstacles and future opportunities.*, Florence, Italy, 2004. System Safety Society.

[63] S.C. Kleene. *Introduction to Metamathematics.* Van Nostrand, Princeton, 1964.

[64] J. C. Knight. Safety critical systems: challenges and directions. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 547–550, New York, NY, USA, 2002. ACM.

[65] M. S. Koons and J. D. Sephton. System Safety Issues and Differences Faced on the International Front. In *20th International System Safety Conference*, 2002.

[66] G. Lee, J. Howard, and P. Anderson. Safety-Critical Requirements Specification and Analysis Using SpecTRM. In *2nd Meeting of the US Software System Safety Working Group*, February 2002.

[67] E. Letier and A. van Lamsweerde. High Assurance Requires Goal Orientation. In *RHAS'02, International Workshop on Requirements for High Assurance Systems*, Essen, Germany, 2002.

[68] Emmanuel Letier and A. van Lamsweerde. Agent-based tactics for goal-oriented requirements elaboration. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 83–93, New York, NY, USA, 2002. ACM.

[69] N. Leveson. *Safeware: system safety and computers*. Addison-Wesley, Reading, Mass., 1995.

[70] N. Leveson. Completeness in formal specification language design for process-control systems. *Proceedings of the third workshop on Formal methods in software practice 2000, Portland, Oregon. ACM Press*, 2000.

[71] N. Leveson. Intent Specifications: An Approach to Building Human-Centered Specifications. *IEEE Transactions on Software Engineering*, Vol. 26(1):15–35, 2000.

[72] N. Leveson. The Role of Software in Recent Aerospace Accidents. In *19th International System Safety Conference*, Huntsville, Alabama, USA, 2001. System Safety Society.

[73] N. Leveson, M. P. E. Heimdahl, and J. D. Reese. Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future. *SIGSOFT Foundations of Software Engineering*, 1999.

[74] Z. Li. *Progressing Problems from Requirements to Specifications in Problem Frames*. PhD thesis, Department of Computer Faculty of Mathematics and Computing ,The Open University, 2007.

[75] Z. Li. Progressing Problems from Requirements to Specifications in Problem Frames. In *IWAAPF '08*. ACM, 2008.

[76] O. Lisagor, J. McDermid, and D. Pumfrey. Safety Analysis of Software Architectures - Lightweight PSSA. In *22nd International System Safety Conference*, Providence, Rhode Island, USA, 2004. System Safety Society.

[77] O. Lisagor, J. McDermid, and D. Pumfrey. Towards a Practicable Process for Automated Safety Analysis. In *24th International System Safety Conference*, Albuquerque, New Mexico, USA, 2006. System Safety Society.

[78] J. Losee. *A Historical introduction to the philosophy of science.* 3rd Edition, Oxford University Press, Oxford, England, 1993.

[79] R. R. Lutz. Analysing Software Requirements Errors in Safety-Critical Embedded Systems. In *IEEE International Symposium Requirements Engineering*, San Diego, California, 1993.

[80] R. R. Lutz. Software engineering for safety: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 213–226, New York, NY, USA, 2000. ACM.

[81] R. R. Lutz and H.-Y. Shaw. Applying adaptive safety analysis techniques. In *10th International Symposium on Software Reliability Engineering*, San Diego, California, 1999.

[82] Tom Maibaum. Mathematical Foundations of Software Engineering: a roadmap. In *ICSE 2000*, King's College, London, 2000.

[83] D. Mannering. Implementable Requirements in Problem Orientation. In *IWAAPF '08*. ACM, 2008.

[84] D. Mannering and A Fry. Process Safety Review. IPDP Guidance Document Rev. 2, GD UK Ltd., December 2006.

[85] D. Mannering, J. G. Hall, and L. Rapanotti. Safety process improvement: Early analysis and justification. In *Proceedings of the IET Second International Conference on System Safety 2007*. IET, 2007.

[86] D. Mannering, J. G. Hall, and L. Rapanotti. Safety process improvement with pose and alloy. In *Proceedings of The 26th International Conference on Computer Safety, Reliability and Security (Safecomp 2007)*. Springer, 2007.

[87] D. Mannering, J. G. Hall, and L. Rapanotti. Towards Normal Design for Safety-Critical Systems. In *Proceedings of FASE 2007*, pages 398–411, 2007.

[88] D. Mannering, J. G. Hall, and L. Rapanotti. Safety Process Improvement: Using POSE and Alloy. In *Safety System Symposium SSS'08*, pages 398–411, Bristol, England, February 2008.

[89] P. A. Martino and C. Muniak. The Role of System Safety Engineering in Product Safety. In *20th International System Safety Conference*, pages 439–447, Denver, Colorado, USA, 2002. System Safety Society.

[90] J. A. McDermid. Software Hazard and Safety Analysis. In *FTRTFT 2002, LNCS 2469*, pages 22–34, Berlin, September 2002 2002. Springer-Verlag.

[91] J. A. McDermid. Trends in System Safety: A European View? In *7th Australian Workshop Conference on Safety Critical Systems and Software*, 2002.

[92] J. A. McDermid and T. P. Kelly. Safety and Hazard Analysis Course. Technical report, York, 1999.

[93] J. A. McDermid and D. Pumfrey. Software Safety: Why is there no Consensus? In *19th International System Safety Conference*, Huntsville, Alabama, USA, 2001. System Safety Society.

[94] Jonathan Moffett, Jon Hall, Andrew Coombes, and John McDermid. A model for a causal logic for requirements engineering . *Requirements Engineering*, 1(1):27–46, March 1996.

[95] J. Murdoch and J. A. McDermid. Failure Modes and Effects Analysis (FMEA) and Systematic Design. In *19th International System Safety Conference*, Huntsville, Alabama, USA, 2001. System Safety Society.

[96] B. Nuseibeh and S. Easterbrook. Requirements engineering: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 35–46, New York, NY, USA, 2000. ACM.

[97] F. Ortmeier, A. Thums, G. Schellhorn, and W. Reif. Combining Formal Methods and Safety Analysis – The ForMoSA Approach. In *Integration of Software Specification Techniques for Applications in Engineering*. Springer LNCS 3147, 2004.

[98] E. N. Overton. System Safety Analysis of Safety Critical Software and the Human User. In *19th International System Safety Conference*, Huntsville, Alabama, USA, 2001. System Safety Society.

[99] S. Owre, S. Rajan, J. Rushby, N. Shankar, and M. Srivas. *PVS: Combining specification, proof checking, and model checking*, volume 1102. Springer Berlin, 1996.

[100] David L. Parnas and J. Madey. Functional documents for computer systems. *Science of Computer Programming*, 25(25):41–61, 1995.

[101] T. Peikenkamp, A. Cavallo, L. Valacca, E. Bŏde, M. Pretzer, and E. M. Hahn. Towards a Unified Model-Based Safety Assessment. In *SAFECOMP 2006, LNCS 4166*, pages 275–288, Berlin, 2006. Springer-Verlag.

[102] D. Pumfrey. *The Principled Design of Computer System Safety Analyses.* PhD thesis, Department of Computer Science,University of York, 2000.

[103] L. Rapanotti, J. G. Hall, M. A. Jackson, and B. Nuseibeh. Architecture-driven Problem Decomposition. In *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International*, pages 80–89, Washington, DC, USA, 2004. IEEE Computer Society.

[104] V. Ratan, K. Partridge, J. D. Reese, and N. Leveson. Safety Analysis Tools for Requirements Specifications. In *COMPASS'96, 11th Annual Conference on Computer Assurance*, 1996.

[105] F. Redmill. An Introduction to the Safety Standard IEC 61508. *Journal of the System Safety Society*, 35(1), 1999.

[106] F. Redmill, M. Chudleigh, and J. Catmur. *System Safety: HAZOP and Software HAZOP.* John Wiley & Sons, 1999.

[107] Jon Damon Reese and N. Leveson. Software deviation analysis. In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, pages 250–260, New York, NY, USA, 1997. ACM.

[108] RTCA/DO-178B. Software Considerations in Airborne Systems and Equipment Certification. Technical report, December 1 1992.

[109] SAE. ARP4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. Technical report, 1996.

[110] G. Schedl, A. Gerstinger, and A. Ronach. Safety Management Approach for the Voice Communication System According to Eurocontrol's Air Navigation System Safety Assessment Methodology. In *19th International System Safety Conference*, Huntsville, Alabama, USA, 2001. System Safety Society.

[111] S. Schneider. *The B-method: An Introduction.* Palgrave, 2001.

[112] S. G. Schoolcraft, V. H. Guthrie, and P. B. Parikh. Software Safety Analysis: Inductive versus Deductive Methods. In *18th International System Safety Conference*, Ft. Worth, Texas, USA, 2000. System Safety Society.

[113] R. Seater and D. Jackson. Problem frame transformations: deriving specifications from requirements. In *IWAAPF '06*, pages 71–80, New York, NY, USA, 2006. ACM.

[114] R. Seater, D. Jackson, and R. Gheyi. Requirement progression in problem frames: deriving specifications from requirements. *Requir. Eng.*, 12(2):77–102, 2007.

[115] T. Sorell. *Descartes.* Oxford University Press, Oxford, England, 1987.

[116] J. M. Spivey. *The Z-Notation - A Reference Manual, 2nd ed.* Prentice Hall, 1992.

[117] E. A. Strunk and J. C. Knight. The essential synthesis of problem frames and assurance cases. In *IWAAPF '06*, pages 81–86. ACM, 2006.

[118] Systems-Group. Writing Good Requirements and the Requirements Management Process: DOORS Requirements Process. Module F E195F, General Dynamics United Kingdom Limited, June 2009.

[119] E. Troubitsyna. Specifying Safety-Related Hazards Formally. In *17th International System Safety Conference*, Orlando, Florida, USA, 1999. System Safety Society.

[120] W. M. Turski. And no philosopher's stone, either. In *Information Processing 86, Proceedings of the IFIP 10th World Computer Congress*, pages 1077–1080, Amsterdam, 1986. North-Holland.

[121] W. M. Turski and T. S. E. Maibaum. *The Specification of Computer Programs*. Adison-Wesley, International Computer Science Series, 1987.

[122] Wladyslaw M. Turski. It was fun. *Inf. Process. Lett.*, 88(1-2):7–12, 2003.

[123] UK-MoD. Safety Management Requirements for Defence Systems Part 1 Requirements. Defence Standard 00-56 Issue 2, MoD, 13 December 1996.

[124] UK-MoD. HAZOP Studies on Systems Containing Programmable Electronics; Part I and II. Defence Standard 00-58 Issue 2, MoD, 19 May 2000.

[125] UK-MoD. Safety Management Requirements for Defence Systems Part 1 Requirements. Defence Standard 00-56 Issue 4, MoD, 1 June 2007.

[126] A. van Lamsweerde. Formal specification: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 147–159, New York, NY, USA, 2000. ACM.

[127] A. van Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective. In *22nd ICSE'00*, Limerick, 2000.

[128] A. van Lamsweerde, R. Darimont, and P. Massonet. Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt. *RE*, 00, 1995.

[129] A. van Lamsweerde and L. Willemet. Inferring declarative requirements specifications from operational scenarios. *IEEE Trans. on Software Engineering*, 24(12):1089–1114, 1995.

[130] W. Vesely, F. Goldberg, N. Roberts, and D. Haasl. *Fault Tree Handbook*, volume NUREG-0492. U.S. Nuclear Regulatory Commission, 1981.

[131] W. G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. The Johns Hopkins University Press, 1990.

[132] R. Weaver, J. Fenn, and T. P. Kelly. A Pragmatic Approach to Reasoning about the Assurance of Safety Arguments. In *8th Australian Workshop on Safety Critical Systems and Software (SCS'03)*, Canberra, Australia, 2003. Australian Computer Society.

[133] P. Zave and M. A. Jackson. Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology*, VI(1):1–30, 1997.